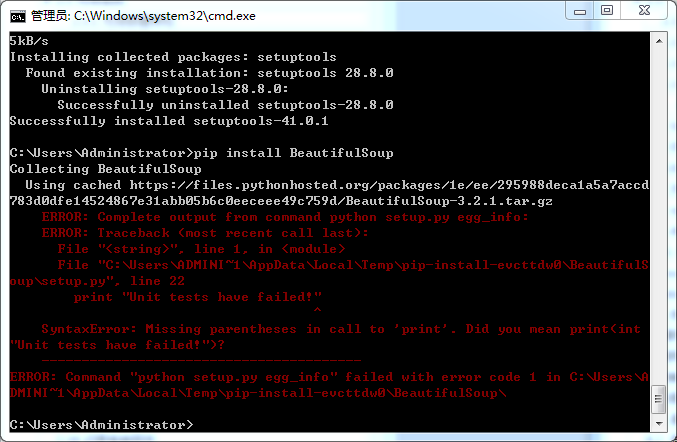
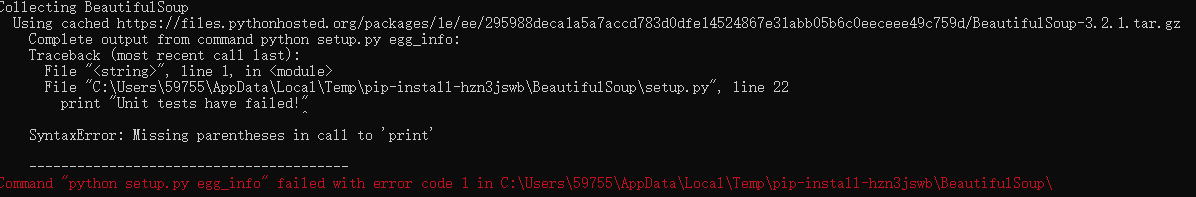
# 安装beautifulsoup失败

## missing parentheses in call to



# [用pip install BeautifulSoup安装BeautifulSoup时出现：SyntaxError: Missing parentheses in call to 'print'](https://www.cnblogs.com/feng525/p/9877084.html)



当出现这种情况时，是因为你所使用的python为3以上版本，而BeautifulSoup不支持高版本的Python,所以解决办法是：

将pip install BeautifulSoup 改为

    pip install BeautifulSoup4

便可安装成功

# 安装docker

## 安装 postgressql

执行命令

yum install postgresql-server

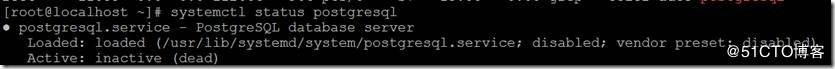
[clip_image001](https://s1.51cto.com/images/blog/201803/25/22d73a3c4d88aad2c1dc476e1d90dc61.png?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

yum install postgresql-contrib

[clip_image002](https://s1.51cto.com/images/blog/201803/25/e4e94c4f299c1145a65a2e956b74f78d.png?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

安装完成后，检查postgresql的服务状态

systemctl status postgresql

[](https://s1.51cto.com/images/blog/201803/25/7d83d5955be591b6d9a686a5bc19507e.jpg?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

服务未启动

systemctl start postgresql启动服务

执行过程有可能会报错

Job for postgresql.service failed because the control process exited with error code. See "systemctl status postgresql.service" and "journalctl -xe" for details.

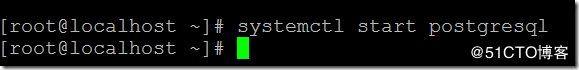
[clip_image006](https://s1.51cto.com/images/blog/201803/25/7ce9308eddcc366ef18685125bde7df7.jpg?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

安装postgresql后需要初始化数据库

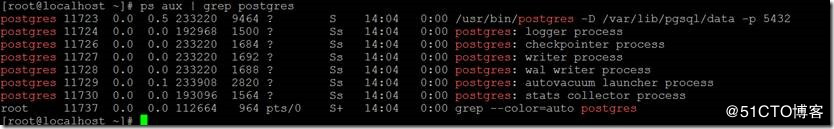
执行postgresql-setup initdb

[clip_image007](https://s1.51cto.com/images/blog/201803/25/182d8ac124394c5f344e0754f9546179.png?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

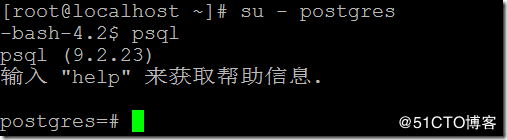
再次启动postgresql服务

[](https://s1.51cto.com/images/blog/201803/25/cb2b964af76e50f59f323d7c42977f58.png?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

用ps命令查看postgresql进程信息

[](https://s1.51cto.com/images/blog/201803/25/64334e762748bb7677908be1e8740149.jpg?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

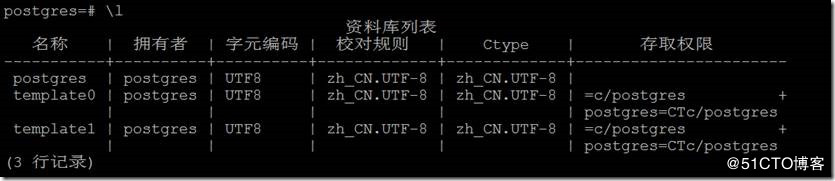
用su – postgres切换到postgresql用户执行psql的操作

[](https://s1.51cto.com/images/blog/201803/25/714714d41ba6893af33df848b1d903eb.png?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

psql -U postgres 登录数据库，执行后提示符变为 ‘postgres=#’

ALTER USER postgres WITH PASSWORD ‘postgres’

使用\l查看postgresql的所有数据库信息

[](https://s1.51cto.com/images/blog/201803/25/42ae723e6c9098a9f662c52253748bd5.jpg?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

到此可以看到postgresql已正确安装。为了以后每次开机可以启动postgresql，执行如下命令

[clip_image015](https://s1.51cto.com/images/blog/201803/25/e350b2a3c5940fc30bdf58ecb516d026.jpg?x-oss-process=image/watermark,size_16,text_QDUxQ1RP5Y2a5a6i,color_FFFFFF,t_100,g_se,x_10,y_10,shadow_90,type_ZmFuZ3poZW5naGVpdGk=)

到此postgresql基本的安装和配置完成。

PostgreSQL 安装完成后，会建立一下‘postgres’用户，用于执行PostgreSQL，数据库中也会建立一个’postgres’用户，默认密码为自动生成，需要在系统中改一下。

修改postgres用户密码：

先切换到root   
然后passwd postgress   
接着输入两遍新密码

修改postgres数据库管理员密码：

su - postgres 切换用户，执行后提示符会变为 ‘-bash-4.2$’，切换为UNIX风格的bash   
psql -U postgres 登录数据库，执行后提示符变为 ‘postgres=#’   
ALTER USER postgres WITH PASSWORD ‘abc123’ 设置postgres用户密码   
\q 退出数据库

切换到root配置一下远程连接。   
开启远程访问

vim /var/lib/pgsql/data/postgresql.conf   
修改#listen\_addresses = ‘localhost’ 为 listen\_addresses=’\*’   
当然，此处‘\*’也可以改为任何你想开放的服务器IP

信任远程连接

vim /var/lib/pgsql/data/pg\_hba.conf   
修改如下内容，信任指定服务器连接   
# IPv4 local connections:   
host all all 127.0.0.1/32 trust   
两处修改：

1.postgresql.conf

listen\_addresses项值设定为“\*”，在9.0 Windows版中，该项配置已经是“\*”无需修改。

2.pg\_hba.conf

host all all 127.0.0.1/32 md5行下添加以下配置

host    all    all    0.0.0.0/0    md5

如果不希望允许所有IP远程访问，则可以将上述配置项中的0.0.0.0设定为特定的IP值。

Linux下可通过find . -name "pg\_hba.conf" 查找文件位置

window下 直接安装目录下 data/ 例如：C:\Program Files\PostgreSQL\9.3\data

远程连接配置完成，由于系统原因，还需要在防火墙中打开相应的端口。

打开防火墙   
CentOS 防火墙中内置了PostgreSQL服务，配置文件位置在/usr/lib/firewalld/services/postgresql.xml，我们只需以服务方式将PostgreSQL服务开放即可。

firewall-cmd –add-service=postgresql –permanent 开放postgresql服务   
firewall-cmd –reload 重载防火墙

最后一步，不能忘记的，是重启数据库服务，使配置生效。

重启PostgreSQL数据服务

**至此，PostgreSQL 10 在CentOS 7上完成基本安装和配置。**

下面是postgersSQL的使用，在使用postgres的时候要在postgres用户下使用。

createdb mydb

执行这句的时候回出现很多问题，具体见：[doc](https://www.postgresql.org/docs/10/static/tutorial-createdb.html)   
下面是常见的一种错误：

createdb: could not connect to database postgres: FATAL: role “joe” does not exist   
这个是因为没有以postgres用户执行的原因。

执行SQL语句：

psql mydb

之后就可以像普通关系型数据库一样执行SQL语句了，具体参看[doc](https://www.postgresql.org/docs/10/static/tutorial-accessdb.html)

## 安装dockercompose

**Docker-Compose是一个部署多个容器的简单但是非常必要的工具.**

安装Docker-Compose之前，请先安装 python-pip

## 安装 python-pip

### 1、首先检查linux有没有安装python-pip包，终端执行 pip -V

|  |  |
| --- | --- |
| 1  2 | [root@vm-50-151 network-scripts]# pip -V  -bash: pip: command not found |

### 2、没有python-pip包就执行命令 yum -y install epel-release

### 3、执行成功之后，再次执行yum -y install python-pip

### 4、对安装好的pip进行升级 pip install --upgrade pip

　　至此，pip安装好了，执行pip -V 再次检查pip环境。

## 安装Docker-Compose

　　1.终端执行：pip install docker-compose

|  |  |
| --- | --- |
| 1 | 报错：ReadTimeoutError: HTTPSConnectionPool(host='pypi.python.org', port=443): Read timed out |

　　解决：pip --default-timeout=200 install -U docker-compose

　　检查docker-compose 安装：docker-compose -version

|  |  |
| --- | --- |
| 1  2 | [root@vm-50-151 network-scripts]# docker-compose -version  docker-compose version 1.17.1, build 6d101fb |

**如果报错：**

|  |  |
| --- | --- |
| 1 | pkg\_resources.DistributionNotFound: backports.ssl-match-hostname>=3.5 |

**解决方法**：更新backports.ssl-match-hostname的版本，在终端输入命令  
　　pip install --upgrade backports.ssl\_match\_hostname

## listen tcp 0.0.0.0:5432: bind: address already in use

关闭postgressql服务再build

## ERROR: Couldn't connect to Docker daemon at http+docker://localhost - is it running?

### Solution:

There are two possible reasons for this error message.

The common reason is that the user you are running the command as does not have the permissions to access docker.

You can fix this either by running the command as root using sudo (since root has the permission to access docker) or adding your user to the docker group:

* sudo usermod -a -G docker $USER

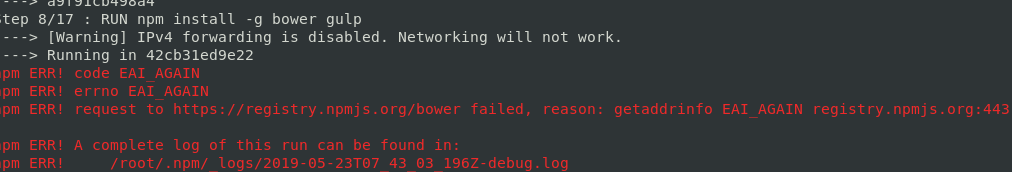
and then logging out and logging back in completely (or restarting the system/server).

The other reason is that you have not started docker. On Ubuntu, you can start it using

* sudo systemctl enable docker # Auto-start on boot
* sudo systemctl start docker # Start right now

## [Warning] IPv4 forwarding is disabled. Networking will not work.

## npm ERR! request to https://registry.npmjs.org/bower failed, reason: getaddrinfo EAI\_AGAIN registry.npmjs.org:443



## AttributeError: 'module' object has no attribute 'GSSException'

[pip uninstall gssapi](https://github.com/paramiko/paramiko/issues/url)

## RROR: Cannot uninstall 'requests'. It is a distutils installed

在配置tensorflow的过程中，通过pip直接安装gpu版本时遇到了这个错误，查了好长时间找不到解决办法，最终摸索出解决办法如下：

我遇到的错误：

Cannot uninstall 'enum34'. It is a distutils installed project and thus we cannot accurately determine which files belong to it which would lead to only a partial uninstall.

解决办法：

找到enum34.egg-info,一般就在usr/lib/python2.7或者其他安装pythonxxx dist-packages等路径里，可以直接在/目录下全局搜索。然后删掉即可。

其他的package遇到这种情况方法也通用，搜索<package name>.egg-info，找到后删除即可

# QProcess实时输出python

在QT程序中我们经常需要与其他的程序间进行交互，而与终端进行交互就是一个例子。在不需要获取返回信息的情况下我们可以直接使用"system()"函数执行，但是当需要获取执行的结果或者需要实时地将结果输出到窗口上时，就比较难办了，需要涉及进程管道等等。而QT提供的QProcess类则大大简化了这一过程。

QProcess是QT提供的与外部程序进行交互的一个类，主要使用到以下的函数：

start(); //启动一个进程

kill(); //关闭启动的外部进程

write(); //向外部进程写入数据

readAllStandardOutput(); //读取外部进程的标准输出

readAllStandardError(); //读取外部进程的错误信息

创建QProcess对象并连接信号与槽

cmd = new QProcess(this);

connect(cmd , SIGNAL(readyReadStandardOutput()) , this , SLOT(on\_readoutput()));

connect(cmd , SIGNAL(readyReadStandardError()) , this , SLOT(on\_readerror()));

实现槽函数

接收到标准输出：

void MainWindow::on\_readoutput()

{

ui->textEdit->append(cmd->readAllStandardOutput().data()); //将输出信息读取到编辑框

}

接收到错误信息：

void MainWindow::on\_readerror()

{

QMessageBox::information(0, "Error", cmd->readAllStandardError().data()); //弹出信息框提示错误信息

}

启动外部程序并写入数据

cmd->start("bash"); //启动终端(Windows下改为cmd)

cmd->waitForStarted(); //等待启动完成

cmd->write("ls\n"); //向终端写入“ls”命令，注意尾部的“\n”不可省略

没有意外的话在编辑框中将看见输出了当前目录下的文件：

同理，可以执行绝大部分在终端中执行的命令，不过有些程序所需要的实现可能不太一样，比如更新软件包，则需执行如下命令：

sudo -S apt-get update

如果不带参数"-S"，则会返回错误信息：“sudo：没有终端存在，且未指定askpass程序”

因此此种方法启动程序的时候如果需要root权限，则需要在"sudo"后加上选项"-S"。

还有的情况就是因为终端启动的时候其实已经帮我们读取了很多环境变量，但是我们在QT程序中直接启动bash的时候这些环境变量尚未被读取，导致一些程序无法启动，报错“未找到命令”。这时需要我们主动去source一遍对应的环境。

善后

QProcess启动的外部程序并不会随着QT程序的关闭而关闭，所以在窗口的销毁函数中加入以下代码，使得窗口销毁时连带终结外部进程。

if(cmd)

{

cmd->close();

cmd->waitForFinished();

}

写在最后

---------------------

作者：字节莫

来源：CSDN

原文：https://blog.csdn.net/weixin\_43690347/article/details/84146821

版权声明：本文为博主原创文章，转载请附上博文链接！

# [Windows下PythonQt编译（vs2015+Qt5.11.2+PythonQt 3.2）探索(4)](https://www.cnblogs.com/jsjrj01/p/9823608.html)

[**Windows下PythonQt编译（vs2015+Qt5.11.2+PythonQt 3.2）探索**](https://www.cnblogs.com/jsjrj01/p/9823608.html)

时间：2018年10月20日

　　笔者最近在做Qt方面的开发工作，需用到脚本程序对程序内部进行扩展，就很自然的想到了PythonQt，下面介绍PythonQt在Windows下的的安装编译心得，水平有限，文笔粗鄙，请谅解。

**本机环境：**

　　1、win7 64 旗舰版

　　2、Qt 5.11.2（MSVC 2015,32 bit）

　　3、Python 3.7.0 （32-bit）

　　4、PythonQt 3.2

**准备工作：**

1、安装最新Qt5.11.2，地址[http://download.qt.io/archive/qt/5.11/](http://download.qt.io/archive/qt/5.11/%20)，直接下载链接。

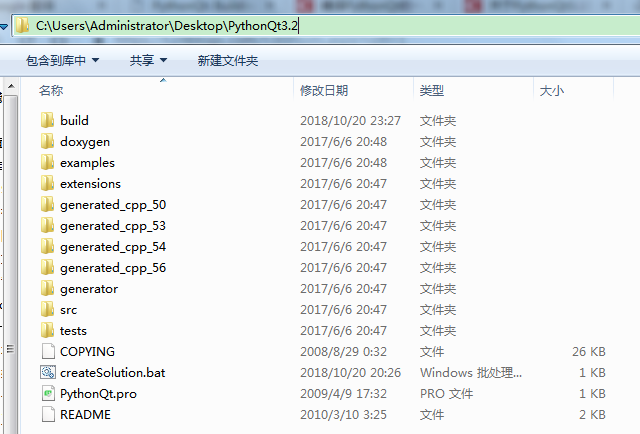
<http://download.qt.io/archive/qt/5.11/5.11.2/qt-opensource-windows-x86-5.11.2.exe>

2、PythonQt是发布在Sourceforge上的一款开源的连接Qt与Python的框架，主页地址[http://pythonqt.sourceforge.net](http://pythonqt.sourceforge.net/) ，目前更新到版本3.2，直接下载链接：<https://sourceforge.net/projects/pythonqt/files/latest/download>，下载后解压。

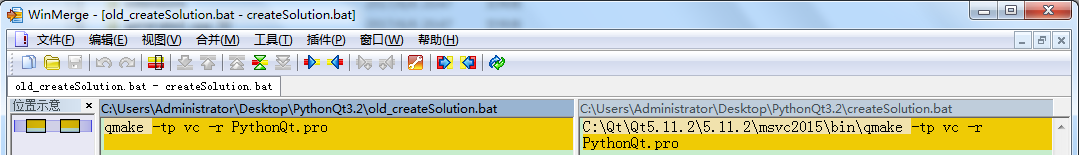
3、下载Python 3.7.0 主页地址<https://www.python.org/downloads/> ，直接下载链接：[https://www.python.org/ftp/python/3.7.0/python-3.7.0.exe](https://www.python.org/ftp/python/3.7.0/python-3.7.0.exe%20)，安装时记得勾选” **Add Python 3.7 to PATH**”以便注册环境变量。

**生成sln工程并编译的过程：**

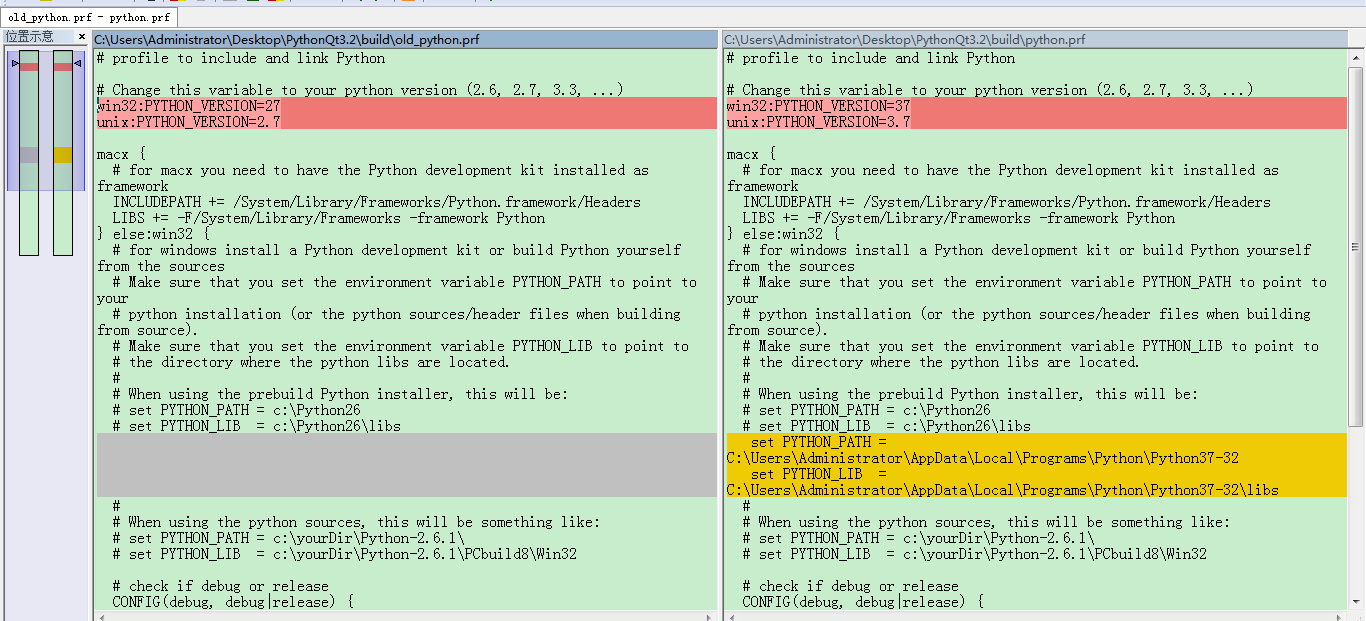
 1、目录结构如下图



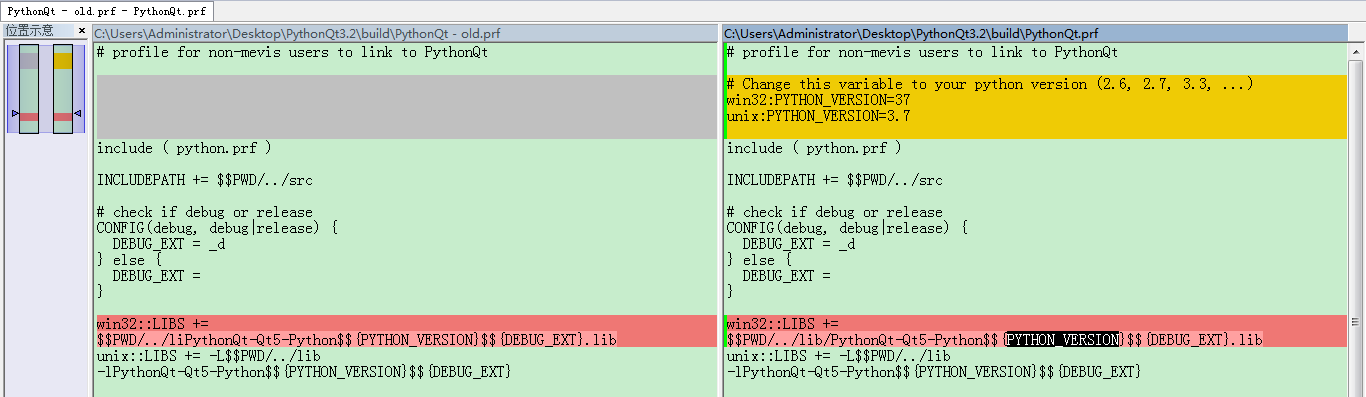
2、将**PythonQt3.2\createSolution.bat**中的内容修改为”**C:\Qt\Qt5.11.2\5.11.2\msvc2015\bin\qmake -tp vc -r PythonQt.pro“**，自己根据Qt安装目录和版本进行调整，这里的qmake一定要是Qt使用的编译器的，我这里是Vs2015的qmake，改为绝对路径是因为我电脑上安装了vc6以及vs多个版本全家桶，环境变量未设置或者相互覆盖，导致混乱，所以没有使用官方文档给出的windows 编译方法PythonQt Building <http://pythonqt.sourceforge.net/Building.html>，修改前后比对如下图所示：



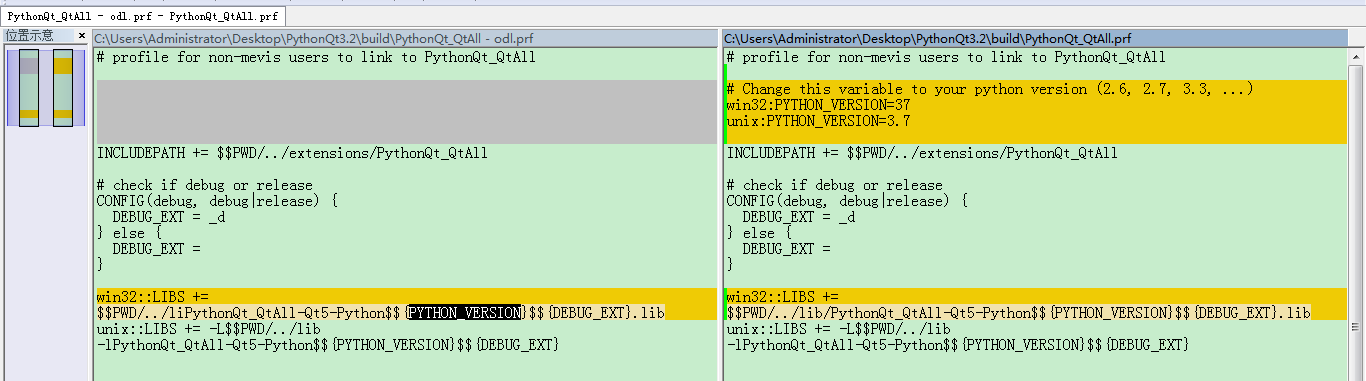
3、修改文件**PythonQt3.2\build\python.prf**的内容，要根据python的版本号和安装路径进行修改，修改前后比对，如下图所示：



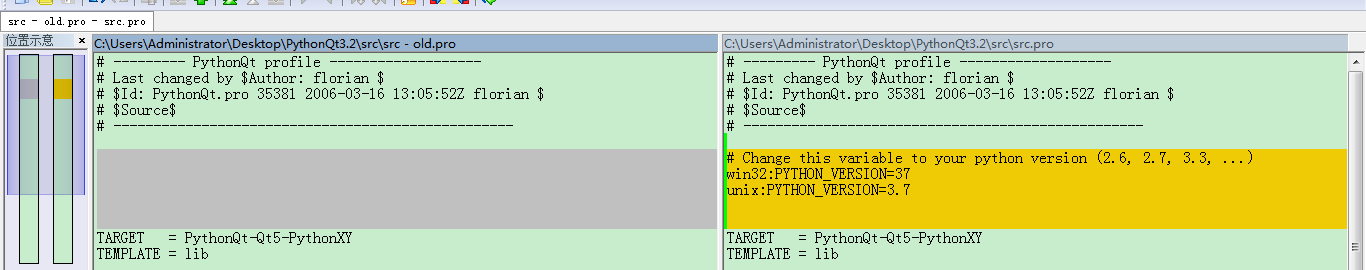
4、修改文件**PythonQt3.2\build\PythonQt.prf**的内容，添加python\_version，为什么python.prf中定义的变量python\_version不好用，我也不清楚。修改前后比对，如下图所示：



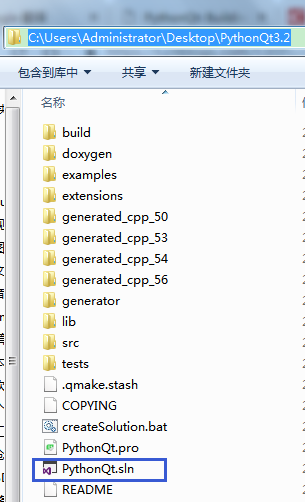
5、修改文件**PythonQt3.2\build\PythonQt\_QtAll.prf**的内容，同样添加python\_version，修改前后比对，如下图所示：



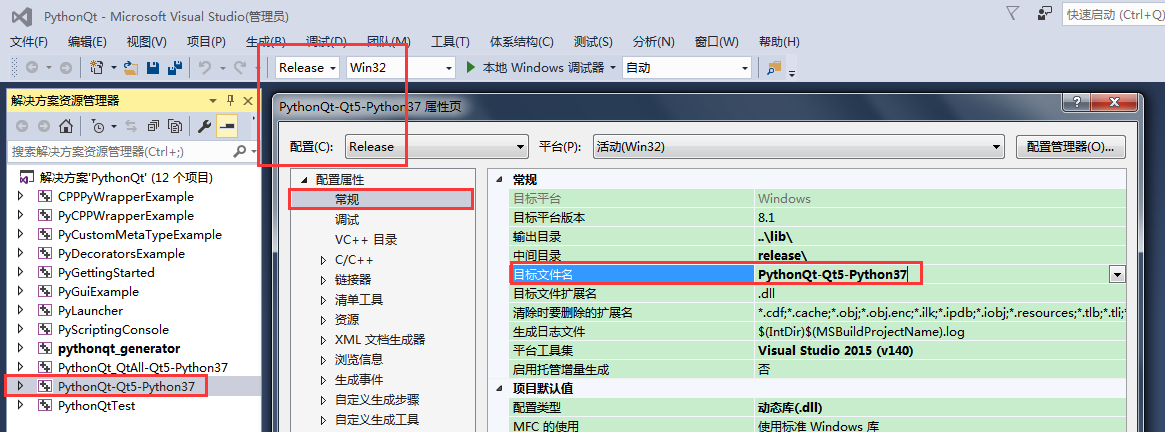
6、修改PythonQt3.2\src\src.pro 文件内容，添加python\_version，修改前后比对，如下图所示：

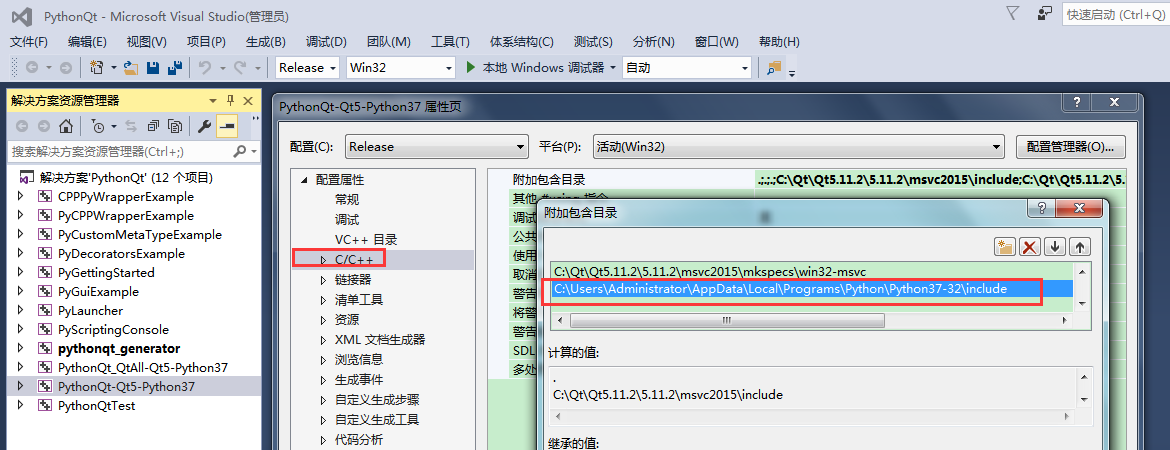


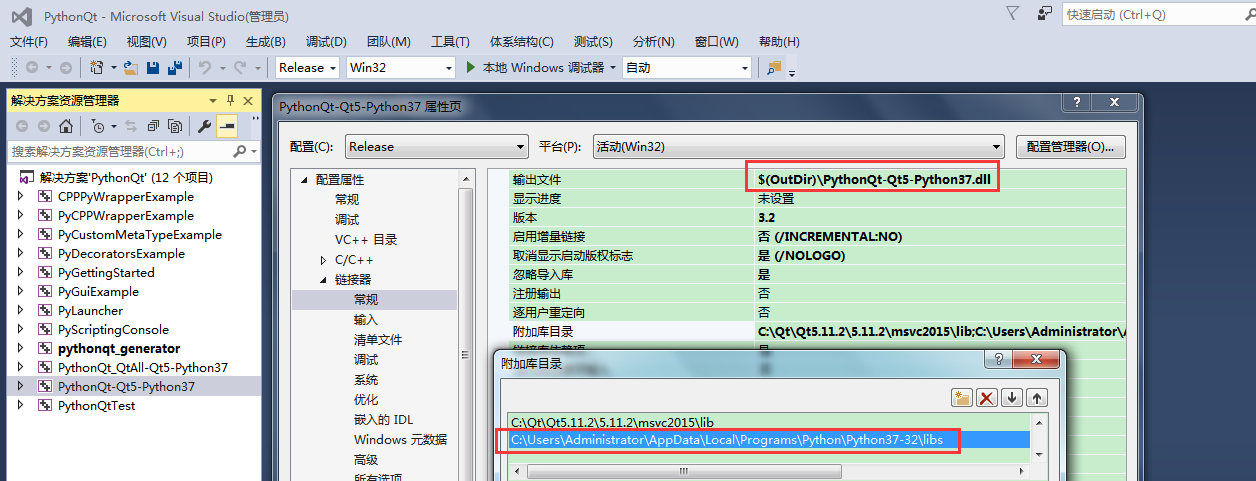
7、右键管理员身份运行**PythonQt3.2\createSolution.bat**（管理员权限不是必须，这里只是习惯），运行结束后，会自动关闭cmd窗口，此时目录中会多出一个PythonQt.sln，目录结构如下图所示：



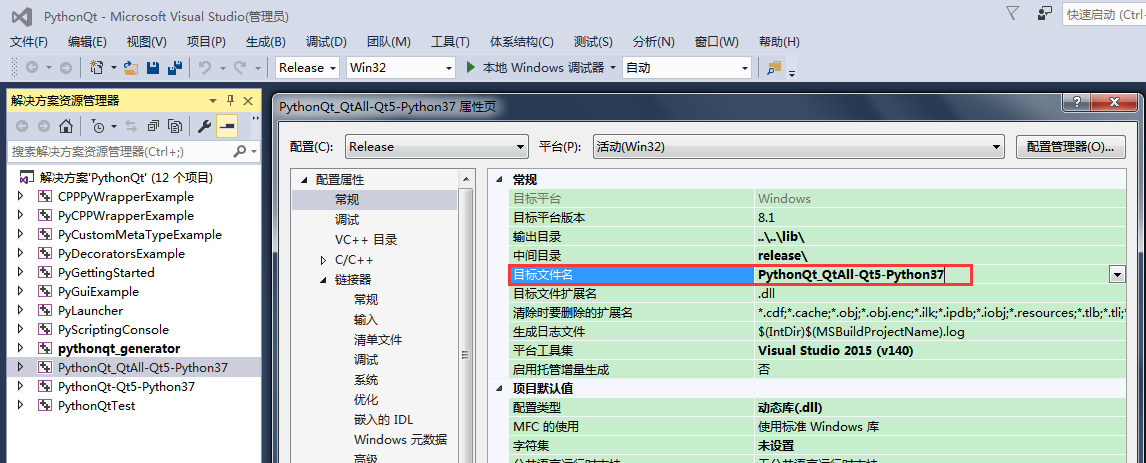
8、使用vs2015打开PythonQt.sln，修改目标文件名为**PythonQt-Qt5-Python37**，根据python的版本和安装路径，设置**PythonQt-Qt5-Python37**工程属性，即添加python的include和lib目录，如下图所示：

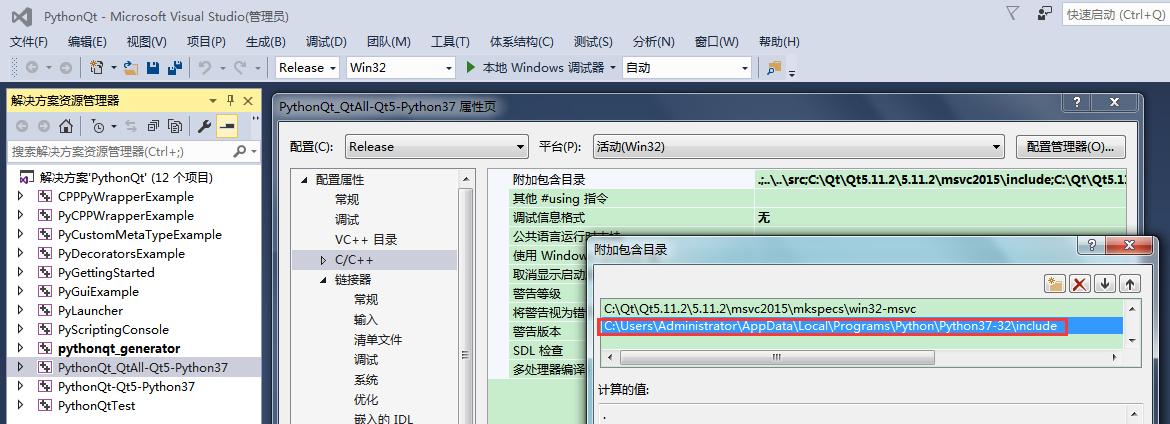


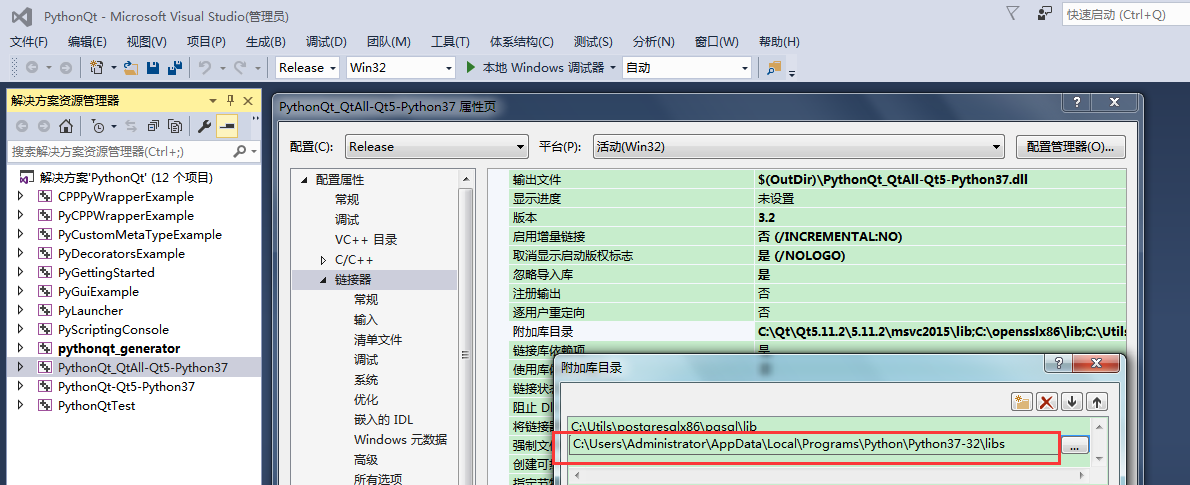


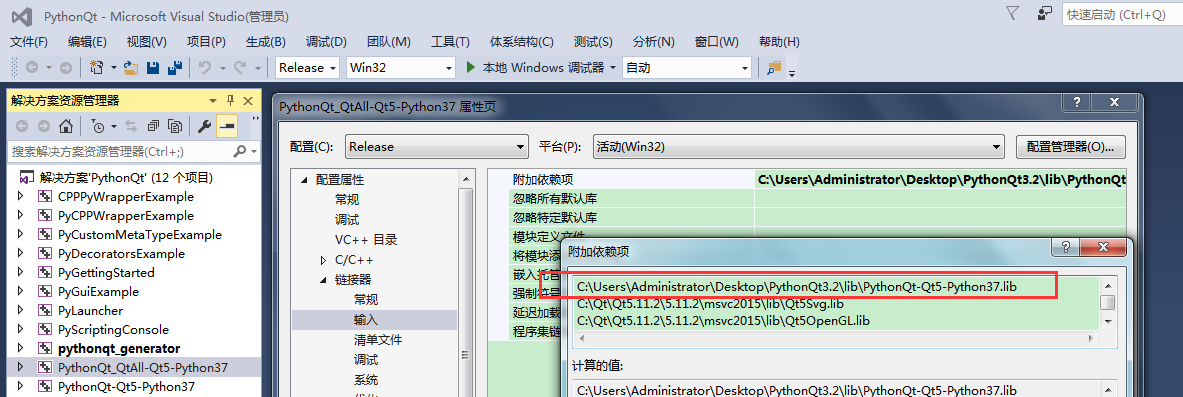


10、根据python的版本和安装路径，设置**PythonQt\_QtAll-Qt5-Python37**工程属性，即添加include和lib目录，如下图所示：







链接器\输入，第一行是下面这样的（批处理自动生成的，这里只是进行核查），说明前面修改prf文件，并且运行批处理的过程基本正确 

11、**右键**，先后编译**①②**，②需要①工程生成的lib和dll，所以编译有先后顺序。

**①PythonQt-Qt5-Python37工程------生成----->PythonQt-Qt5-Python37.dll、PythonQt-Qt5-Python37.lib**

**②PythonQt\_QtAll-Qt5-Python37工程------生成----->PythonQt\_QtAll-Qt5-Python37.lib、PythonQt\_QtAll-Qt5-Python37.dll**

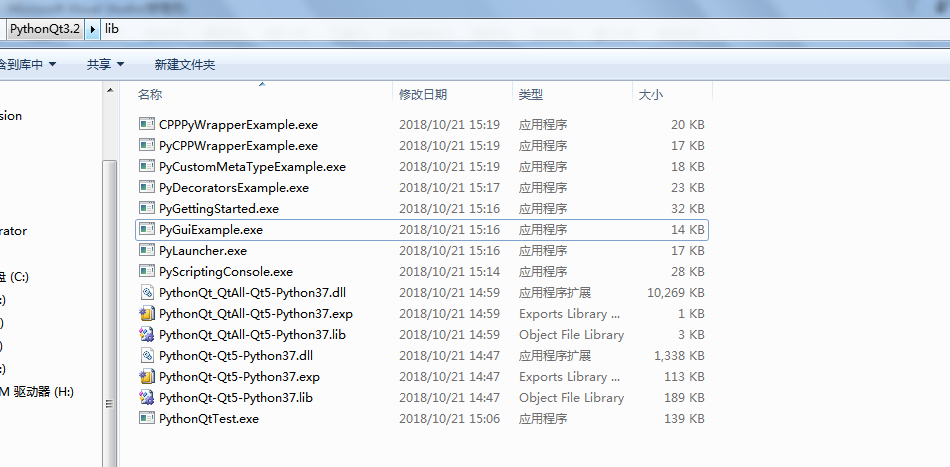
编译ok后，在PythonQt3.2\lib目录下，会生成的两对lib和dll，剩余的工程基本都是依赖它们。

12、剩余的工程基本是些实例，右键编译，如果提示：

　　12.1 fatal error C1083: 无法打开包括文件: “Python.h”: No such file or directory，则添加python的include目录到 附加包含目录（方法和上面的类似）

　　12.2 LINK : fatal error LNK1104: 无法打开文件“python37.lib”，则添加python的libs目录到 附件库目录（方法和上面的类似）

全部编译成功后，lib文件夹内容：



**小结：**

　　使用[PyThonQt](http://pythonqt.sourceforge.net/index.html)，比直接使用python api进行交互式编程更加的便利，官方介绍翻译——”PythonQt是Qt框架的动态Python绑定。它提供了一种将Python脚本语言嵌入C ++ Qt应用程序的简便方法。PythonQt的重点是将Python嵌入到现有的C ++应用程序中，而不是完全用Python编写整个应用程序。如果您正在寻找一种将Python对象嵌入C ++ / Qt应用程序并通过Python 编写应用程序部分脚本的简单方法， 那么PythonQt就是您的最佳选择！PythonQt是一个稳定的库，它是为了使图像处理和可视化平台MeVisLab可以使用Python脚本而开发的“。

　　linux上使用官方文档中的命令行的方式（文献中有链接），编译会很顺利，但在windows上很费劲，经常因为环境变量或者目录的问题出错，这也是使用sln工程编译，而不是命令行式编译的原因。但要要注意因为python安装使用的二进制包，不是使用源码编译，所以debug工程编译不过去的，才有了上面都是Release。官方推荐的做法是，下载python源码，使用自己电脑上的Qt编译后，在进行上面类似的操作。

　　有2个疑问：**为什么python\_version的变量不好用，为什么每个prf文件中设置的python的include和libs目录失效，而是挨个手动修改sln工程属性**，**如果您了解，望不吝赐教，谢谢**。使用这个方法编译参数会和pythonQt的解压目录写死，编译成功后，挪动文件夹位置重新编译就会失败。

　　本文介绍了，使用createSolution.bat生成vs的sln工程，然后编译的过程。其实直接使用Qt打开PythonQt.pro，添加添加include和lib目录，也是可行的。为了解决上面的疑问，百度google无果，到官方论坛，找到了答案。bulid下的prf文件开发人员笔误和一些选项没设定的关系，因为开发人员只是linux/unix测试，windows没有经过完整测试，感情官方给出的在window上编译的方法其实有误的（文献1）。官方论坛开发人员早了解到，但是下载包这么长时间也不更新，哎，windows后娘省的

比如：

win32::LIBS += $$PWD/../liPythonQt\_QtAll-Qt5-Python$${PYTHON\_VERSION}$${DEBUG\_EXT}.lib

应该是

win32::LIBS += $$PWD/../lib/PythonQt\_QtAll-Qt5-Python$${PYTHON\_VERSION}$${DEBUG\_EXT}.lib

额。。。说是拼写错误

再就是生成的lib dll 有后缀3，导致别的工程无法找到对应的lib，这是因为没有设定CONFIG += skip\_target\_version\_ext，因为在unix上不写没有这个问题。经过修改后，vs编译PythonQt不再那么麻烦，呵呵，但是这篇博文写到这里实在不易，于是决定也给此文加上后缀 - 探索。

## 执行.bat找不到cl

C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin加到环境变量

# QT调用python(直接调用)

Qtconsole工程

INCLUDEPATH+= -I C:\Users\Administrator\AppData\Local\Programs\Python\Python36\**include**

LIBS+=-LC:\Users\Administrator\AppData\Local\Programs\Python\Python36\libs\ -lpython36

#**include** <**QCoreApplication**>

#**include** <**Python**.**h**>

#**include** **"../../../cpp/plib/plib.h"**

**int** **main**(**int** argc, **char** \*argv[])

{

QCoreApplication a(*argc*, argv);

*//进行初始化*

Py\_Initialize();

*//如果初始化失败，返回*

**if**(!Py\_IsInitialized())

{

**hlog**(**"init** **fail"**);

**return** -3;

}

*//加载模块，模块名称为myModule，就是myModule.py文件*

PyObject \*pModule = PyImport\_ImportModule(**"filmComent"**);

*//如果加载失败，则返回*

**if**(!pModule)

{

**hlog**(**"load** **fail"**);

**return** -9;

}

*//* *//加载函数greatFunc*

*//* *PyObject* *\** *pFuncHello* *=* *PyObject\_GetAttrString(pModule,* *"greatFunc");*

*//* *//如果失败则返回*

*//* *if(!pFuncHello)*

*//* *{*

*//* *hlog("call* *fail");*

*//* *return* *-1;*

*//* *}*

*//* *//调用函数*

*//* *PyObject\_CallFunction(pFuncHello,* *NULL);*

*//退出*

Py\_Finalize();

**return** a.exec();

}

# Qt调用python(命令行调用)

缺点,错误不提示

QProcess p(0);

*//* *p.start("cmd",QStringList()<<"/c"<<"ipconfig");*

p.start(**"cmd"**,QStringList()<<**"/c"**<<**"python** **C:/Users/Administrator/Desktop/xlibAux/python/tensorflowtest/filmComent.py"**);

p.waitForStarted();

p.waitForFinished();

QString strTemp=QString::fromLocal8Bit(p.readAllStandardOutput());

**python** **C:/Users/Administrator/Desktop/xlibAux/python/tensorflowtest/filmComent.py**

**这句是直接全路径cmd执行,直接用,注意路径别错,qt不报错**

# 最简单方式解决python matplotlib中文乱码问题

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

plt.plot((1,2,3),(4,5,7))

plt.xlabel('横坐标')

plt.ylabel('纵坐标')

plt.show()

---------------------

作者：Yrish

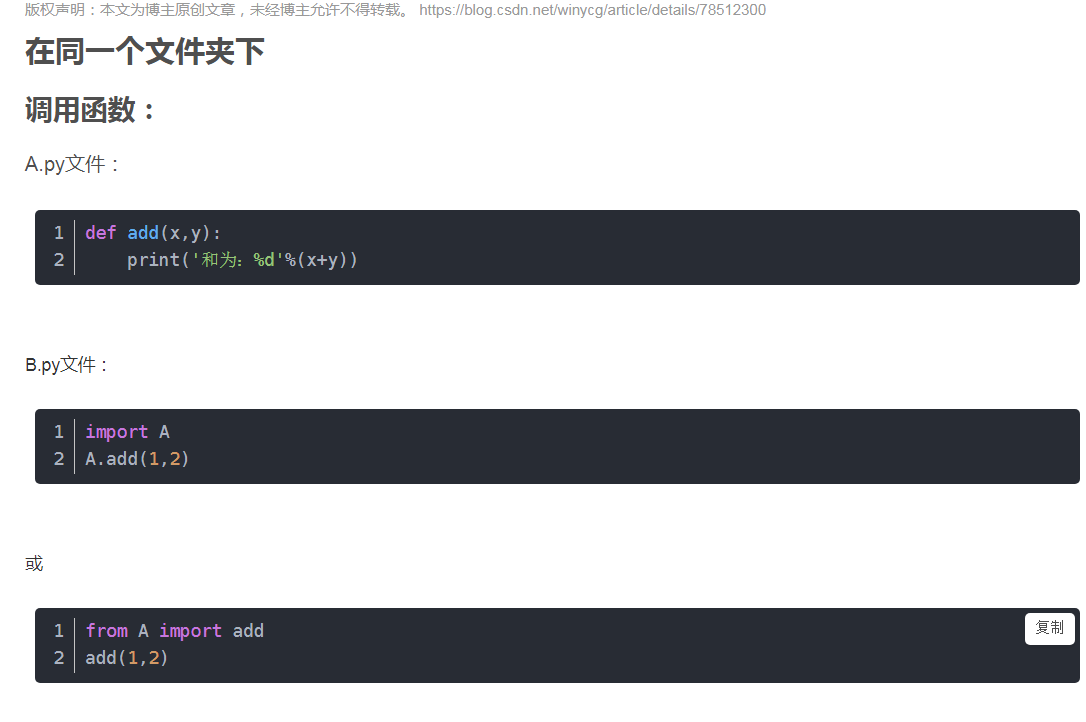
来源：CSDN

原文：https://blog.csdn.net/sinat\_29699167/article/details/80029898

版权声明：本文为博主原创文章，转载请附上博文链接！

# 基本语法

## 调用其它文件中的函数



## 调用其它函数中的类



## 在不同文件夹下



# 人工智能

## 安装keras

Pip install keras

## TensorFlow基本

### 打印矩阵值

**import** tensorflow **as** tf  
*#Initialize the session*sess = tf.InteractiveSession()  
  
matrix1 = tf.constant([[3., 3.]])  
matrix2 = tf.constant([[2.],[2.]])  
product = tf.matmul(matrix1, matrix2)  
  
*#print the product*print(product.eval())  
  
*#close the session to release resources*sess.close()

## Python3.6.3+TensorFlow1.6

Python必须装x64的,且必须3.6,3.5不支持,不然不支持,会报whl is not supported on this platform

1.6 要安装cuda9.0 cudn7.0

安装完成后会提示少cuda,装

Cuda装失败的话,重启再装,会自动配置环境变量,成功就行了

Cudn文件夹放到一个目录,要手动配置环境变量,将文件夹放到任意位置,将bin目录加入环境变量

每次加环境变量要重启

安装tensorflow

升级pip

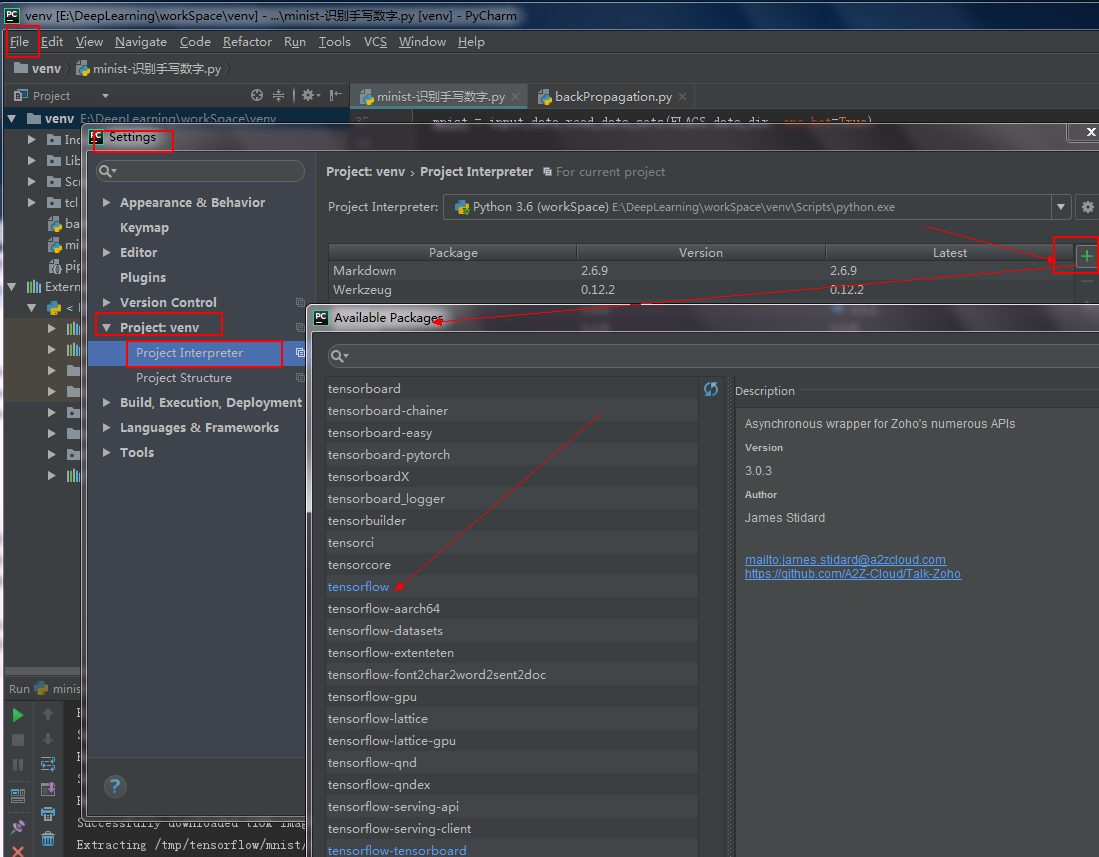
python -m pip install --upgrade pip

pip install \*.whl (命令行安装)

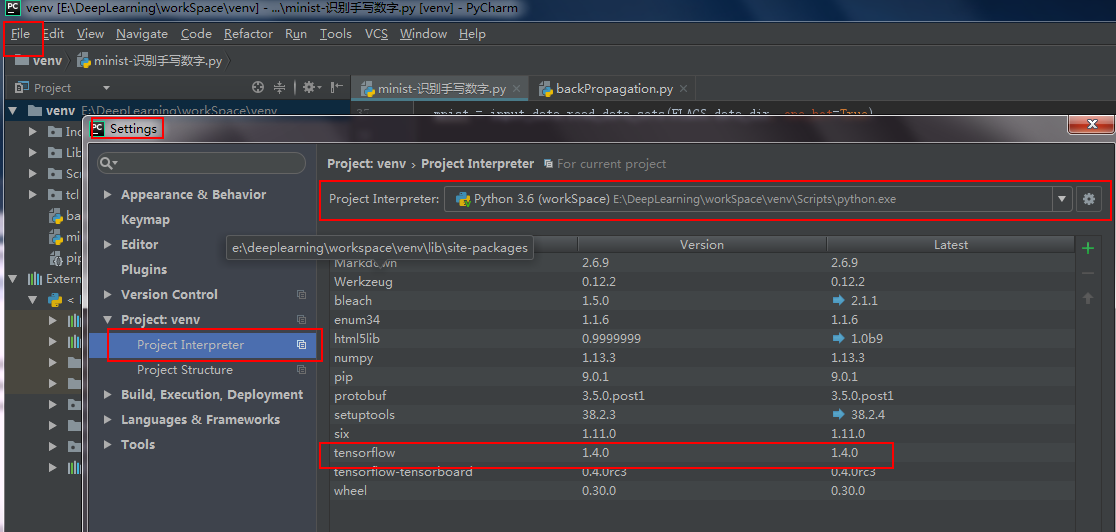
最后的测试:

Pycharm 不能import

我用的pycharm，发现在命令行中，也就是上面的黑框框中可以import tensorflow，可是在pycharm中不行，所以设置一下，在File-->setting中，导入tensorflow这个包，步骤如下图所示：

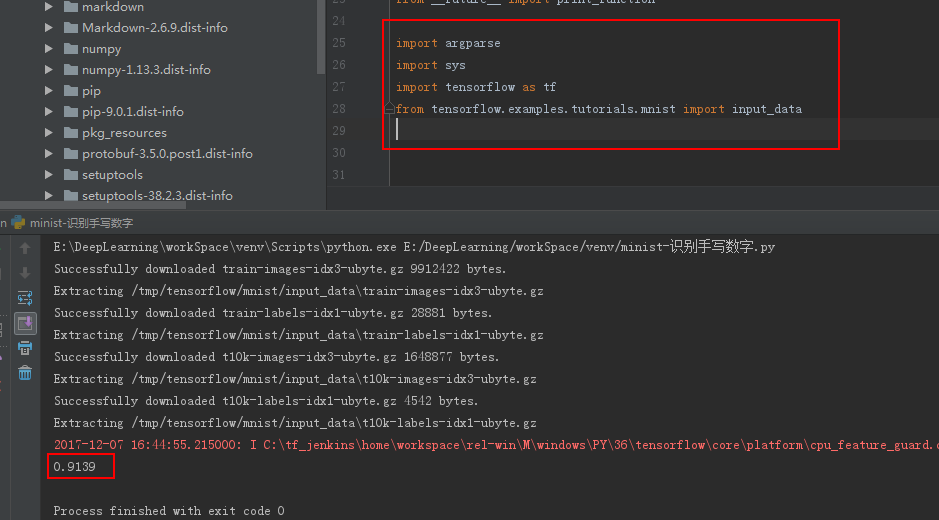


之后会看到如下图所示，你需要的包都有了



 我再pycharm中跑了个例子：<https://github.com/tensorflow/tensorflow/edit/r1.4/tensorflow/examples/tutorials/mnist/mnist_softmax.py>

把官网这个例子拷贝下来就可以run了，这是我run成功的。



分类: [环境搭建相关](http://www.cnblogs.com/catherinezyr/category/1012417.html)

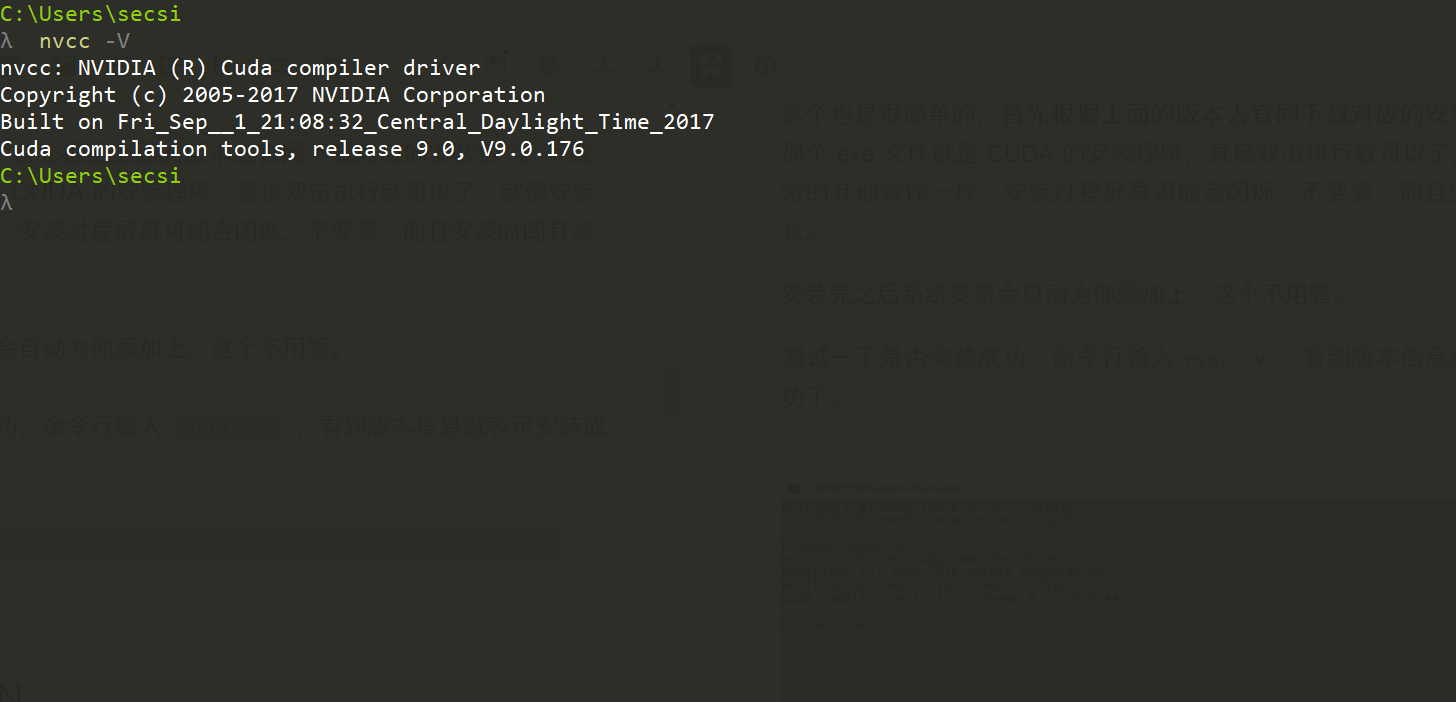
### 安装 CUDA

TensorFlow 1.6：CUDA 9.0

这个也是很简单的，**首先根据上面的版本去**[官网](https://developer.nvidia.com/cuda-toolkit-archive)**下载对应的安装包（~ 1.4 GB）**。下载完那个 exe 文件就是 CUDA 的安装程序，直接双击执行就可以了，就像安装正常的其他软件一样，安装过程屏幕可能会闪烁，不要紧，而且安装时间有点长。

安装完之后系统变量会自动为你添加上，这个不用管。

测试一下是否安装成功，命令行输入 nvcc -V ，看到版本信息就表示安装成功了。



### 安装 cuDNN

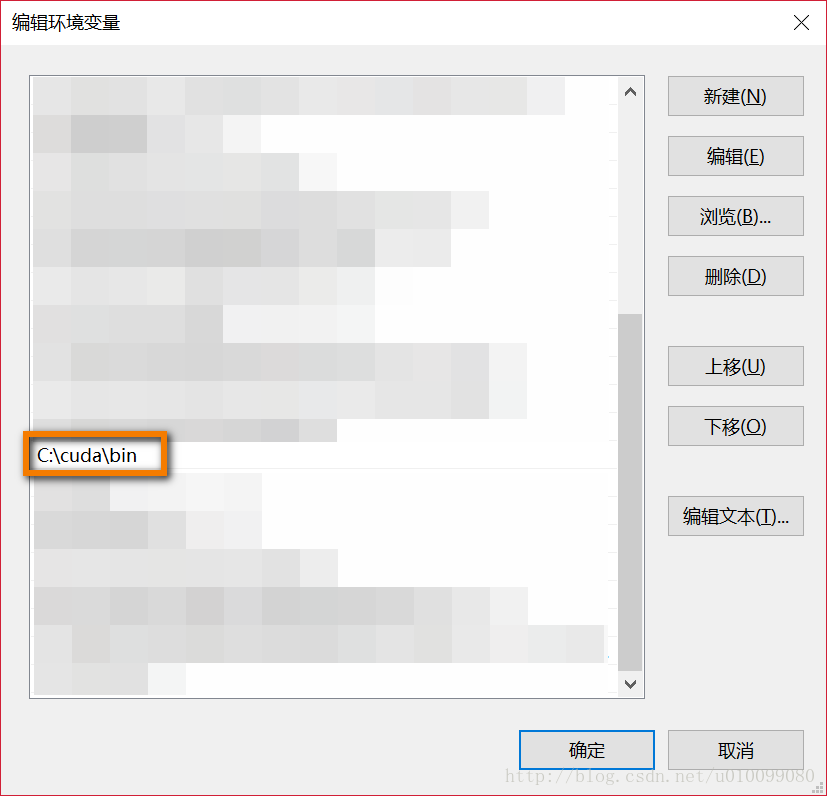
TensorFlow 1.6：cuDNN 7.0

**首先去**[官网](https://developer.nvidia.com/cudnn)**下载对应版本的安装包（~ 173 MB）**。其实这个是一个压缩包，解压放到任何一个目录下就行，然后把其中的bin目录路径添加到Path环境变量里。

比如说我下载解压后放到了下图的 cuda 文件夹：



Path环境变量：



如果你已经安装了 cuDNN 5.0 ，那么升级 cuDNN 的方法可以参考[*这里*](http://blog.csdn.net/u010099080/article/details/57405184) 。

然后 import tensorflow 应该就成功了。

### 测试

### 测试1官网例子

*# import tensorflow as tf  
#  
# a = tf.random\_normal((100, 100))  
# b = tf.random\_normal((100, 500))  
# c = tf.matmul(a, b)  
# sess = tf.InteractiveSession()  
# sess.run(c)  
# print(c)  
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.  
#  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
# http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
# ==============================================================================  
  
"""A very simple MNIST classifier.  
  
See extensive documentation at  
https://www.tensorflow.org/get\_started/mnist/beginners  
"""***from** \_\_future\_\_ **import** absolute\_import  
**from** \_\_future\_\_ **import** division  
**from** \_\_future\_\_ **import** print\_function  
  
**import** argparse  
**import** sys  
  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
  
**import** tensorflow **as** tf  
  
FLAGS = **None  
  
  
def** main(\_):  
 *# Import data* mnist = input\_data.read\_data\_sets(FLAGS.data\_dir, one\_hot=**True**)  
  
 *# Create the model* x = tf.placeholder(tf.float32, [**None**, 784])  
 W = tf.Variable(tf.zeros([784, 10]))  
 b = tf.Variable(tf.zeros([10]))  
 y = tf.matmul(x, W) + b  
  
 *# Define loss and optimizer* y\_ = tf.placeholder(tf.float32, [**None**, 10])  
  
 *# The raw formulation of cross-entropy,  
 #  
 # tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(tf.nn.softmax(y)),  
 # reduction\_indices=[1]))  
 #  
 # can be numerically unstable.  
 #  
 # So here we use tf.nn.softmax\_cross\_entropy\_with\_logits on the raw  
 # outputs of 'y', and then average across the batch.* cross\_entropy = tf.reduce\_mean(  
 tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y))  
 train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)  
  
 sess = tf.InteractiveSession()  
 tf.global\_variables\_initializer().run()  
 *# Train* **for** \_ **in** range(1000):  
 batch\_xs, batch\_ys = mnist.train.next\_batch(100)  
 sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})  
  
 *# Test trained model* correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))  
 accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  
 print(sess.run(accuracy, feed\_dict={x: mnist.test.images,  
 y\_: mnist.test.labels}))  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 parser = argparse.ArgumentParser()  
 parser.add\_argument(**'--data\_dir'**, type=str, default=**'/tmp/tensorflow/mnist/input\_data'**,  
 help=**'Directory for storing input data'**)  
 FLAGS, unparsed = parser.parse\_known\_args()  
 tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)

### 测试2

用一个简单的矩阵乘法测试一下，

import tensorflow as tf

a = tf.random\_normal((100, 100))

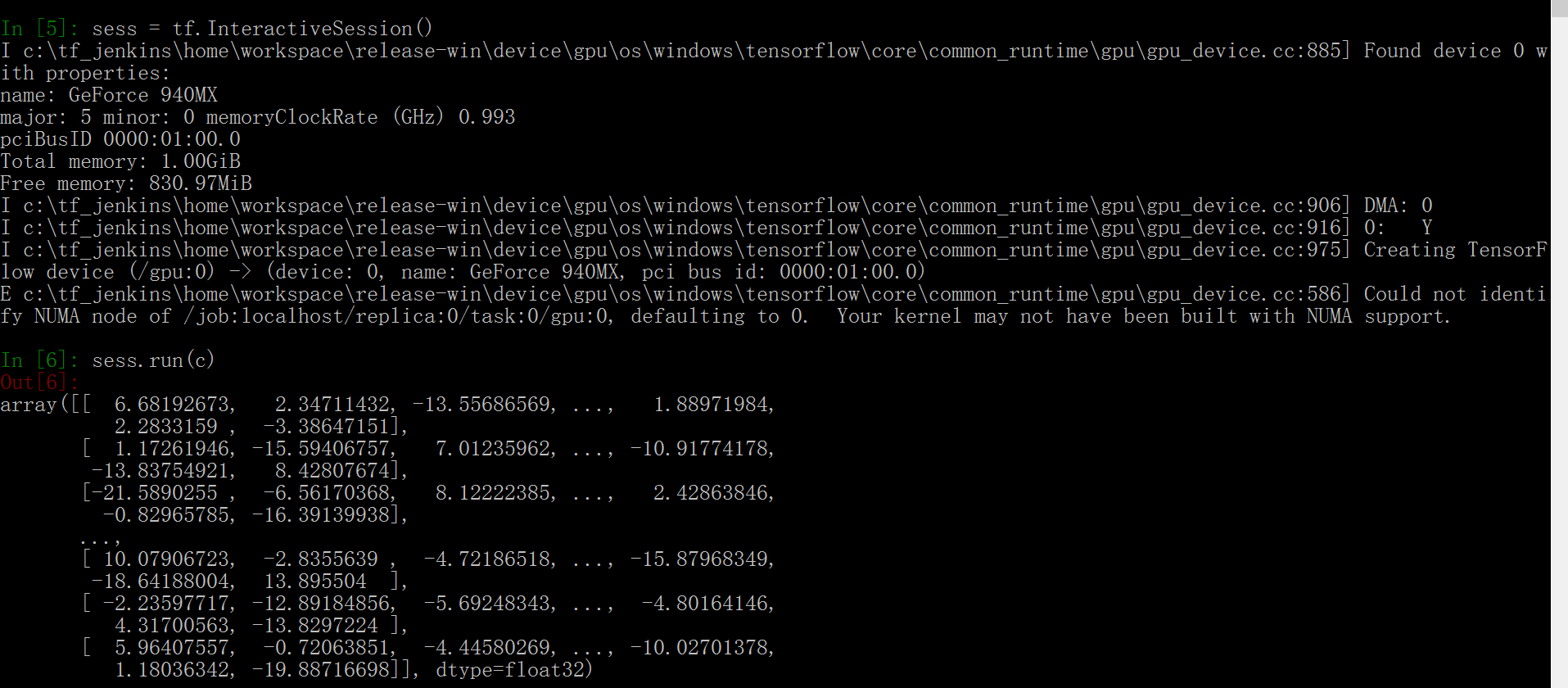
b = tf.random\_normal((100, 500))

c = tf.matmul(a, b)

sess = tf.InteractiveSession()

sess.run(c)

* 1
* 2
* 3
* 4
* 5
* 6
* 7

  
看不清的话可以右键在新标签页打开图片查看大图

可以看到已经识别出我的显卡了（不要吐槽我的渣显卡，穷屌丝一个）。

终于可以在 Windows 上和 TensorFlow 愉快的玩耍了~~~

## 教程1

### 1、安装Python，设置环境变量

下载地址：<https://www.python.org/downloads/windows/>

### 2、下载安装pip

下载地址：<https://pypi.python.org/pypi/pip>

### 3、注意：NVD显卡支持GPU加速的需要安装CUDA和cuDNN；

推荐文章：<http://blog.csdn.net/duwangthefirst/article/details/79078306>

### 4、下载TensorFlow

下载地址：<https://pypi.python.org/pypi/tensorflow#downloads>

error：failed to create process.

python -m pip install tensorflow-1.6.0rc1-cp36-cp36m-win\_amd64.whl

等一会儿，提示安装完成。

  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)  
    100% |████████████████████████████████| 327kB 98kB/s  
Installing collected packages: termcolor, numpy, six, absl-py, wheel, gast, protobuf, grpcio, astor, html5lib, markdown, bleach, werkzeug, tensorboard, tensorflow  
  Running setup.py install for termcolor ... done  
  Running setup.py install for absl-py ... done  
  Running setup.py install for gast ... done  
  Running setup.py install for html5lib ... done

Successfully installed absl-py-0.1.10 astor-0.6.2 bleach-1.5.0 gast-0.2.0 grpcio-1.9.1 html5lib-0.9999999 markdown-2.6.11 numpy-1.14.1 protobuf-3.5.1 six-1.11.0 tensorboard-1.6.0 tensorflow-1.6.0rc1 termcolor-1.1.0 werkzeug-0.14.1 wheel-0.30.0

### 5、pyhton终端验证安装：

>>> import tensorflow as tf  
>>> hello = tf.constant('Hello, TensorFlow!')  
>>> sess = tf.Session()  
>>> print(sess.run(hello))  
b'Hello, TensorFlow!'

参考文章：《**Windows下tensorflow安装 win64+Python3.5**》

<http://blog.csdn.net/btbujhj/article/details/73127753>

## 安装教程2

Preparation：

        工具：windows10+cudn9.0+cudnn7.0+tensorflow1.6.0

        下载：cudn9.0+cudnn7.1，点击进入[云盘](https://pan.baidu.com/s/1pmo9dxHMynQXGby8aLSkkg)可以下载，也可以进入[官方网站](https://developer.nvidia.com/cuda-90-download-archive)进行下载。 密码：vcp1

**Installing：**

相信点开这篇文章的windows10都已经安装好了，不废话，直接正菜，我使用的是Windows10专业版安装，其它版本的没试过。安装方式为tensorflowGPU版本安装。

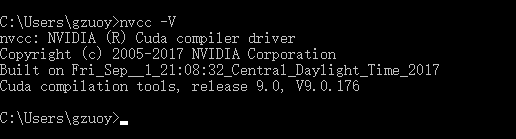
        CPU版本：仅需执行pip3 install --upgrade tesnorflow命令，就可以直接使用了。

 GPU版本：python版本为3.6.4（注：版本必须是3.5以上），使用如下命令安装：

                                                   pip3 install --upgrade tensorflow-gpu

截止到2018-3-8号，tensorflow版本是1.6.0，使用这个命令安装获取到的就是这个版本，安装时间随心随缘，耐心等待就好。等安装好了之后，接下来安装cudn9.0：

        cudn9.0下载下来后，直接安装，直到结束，然后执行命令：nvcc -V，出现下面界面，表示成功：



        安装环境什么不用配置，已经自动配置好了。

        cudnn7.0安装：下载下来后直接解压到一个目录下，再将cudnn文件夹下面的bin目录复制粘贴到path环境变量路径中，就可以直接使用tensorflow了，如：我的解压路径：E:\InsallSoftware\cudnn-9.0-windows10-x64-v7.0\cuda\bin

检验：python环境下，import tensorflow，如果成功，表示安装完成。

**TIPS：**

        1、cudn9.1版本点击进入[云盘](https://pan.baidu.com/s/1346sZNA0vI9CpoO12BcFvg)下载，密码：vwh2，到此结束！谢谢！

        2、cudn9.0版本对应安装cudnn7.0版本，否则运行程序的时候回出现错误

## Python3.6+opencv3.4.0

要先安装numpy,cp36的代表python3.6

# Python3

## 安装

### Windows

### 安装python3

### 安装pycharm

## Linux

源码安装后

**mv -f /usr/bin/python /usr/bin/pythonbk && ln -s /usr/local/bin/python3 /usr/bin/python**

## 错误

### 找不到pylab

pip install matplotlib

### File "<stdin>", line 1

Python3改成了print()

print ("hello world!")

### ***IndentationError:expected an indented block***

在编译时会出现这样的错***IndentationError:expected an indented block***说明此处需要缩进，你只要在出现错误的那一行，按空格或Tab（但不能混用）键缩进就行。

往往有的人会疑问：我根本就没缩进怎么还是错，不对，该缩进的地方就要缩进，不缩进反而会出错，，比如：

***if xxxxxx：***

***（空格）xxxxx***

或者

***def xxxxxx：***

***（空格）xxxxx***

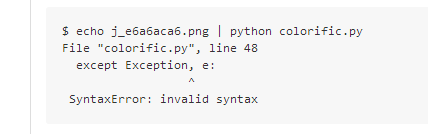
还有

***for xxxxxx：***

***（空格）xxxxx***

一句话 有冒号的下一行往往要缩进，该缩进就缩进

### except Exception, e: SyntaxError: invalid syntax



,改成 as

### Traceback找不到

import traceback

### 升级python3后yum错误

在上一篇升级python的时候的，使用yum时，出现以下错误

**[html]** [view plain](http://blog.csdn.net/harith/article/details/17691137) [copy](http://blog.csdn.net/harith/article/details/17691137)

[在CODE上查看代码片](https://code.csdn.net/snippets/135297)

1. [root@localhost bin]# yum
2. File "/usr/bin/yum", line 30
3. except KeyboardInterrupt, e:
4. ^
5. SyntaxError: invalid syntax
6. [root@localhost bin]#

解决的办法是：

vim /usr/bin/yum

**[html]** [view plain](http://blog.csdn.net/harith/article/details/17691137) [copy](http://blog.csdn.net/harith/article/details/17691137)

[在CODE上查看代码片](https://code.csdn.net/snippets/135297)

1. #!/usr/bin/python
2. import sys
3. try:
4. import yum

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

将上面的语句改为：

**[html]** [view plain](http://blog.csdn.net/harith/article/details/17691137) [copy](http://blog.csdn.net/harith/article/details/17691137)

[在CODE上查看代码片](https://code.csdn.net/snippets/135297)

1. #!/usr/bin/python2.6
2. import sys
3. try:
4. import yum

后面执行yum 就可以使用了

# Pycharm

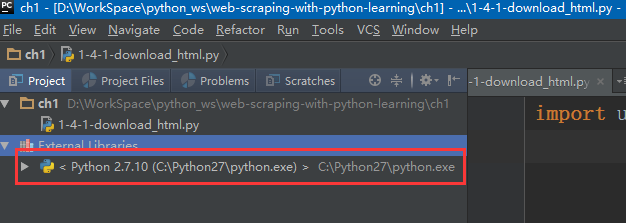
## 更改Python版本

**python2** 和 **python3** 有很大的不同，使用**python2** 编写的程序，如果使用**python3** 就运行不了；使用**python3**编写的程序，如果使用**python2**一样也是不用运行。

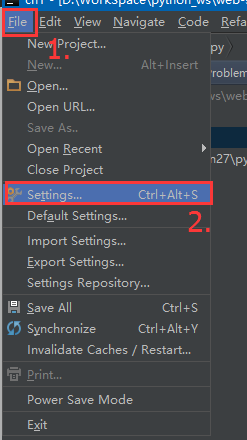
**Q：** 所以，如何在**pyCharm** 软件中，为你的项目设置 **python** 的版本？

**A：**

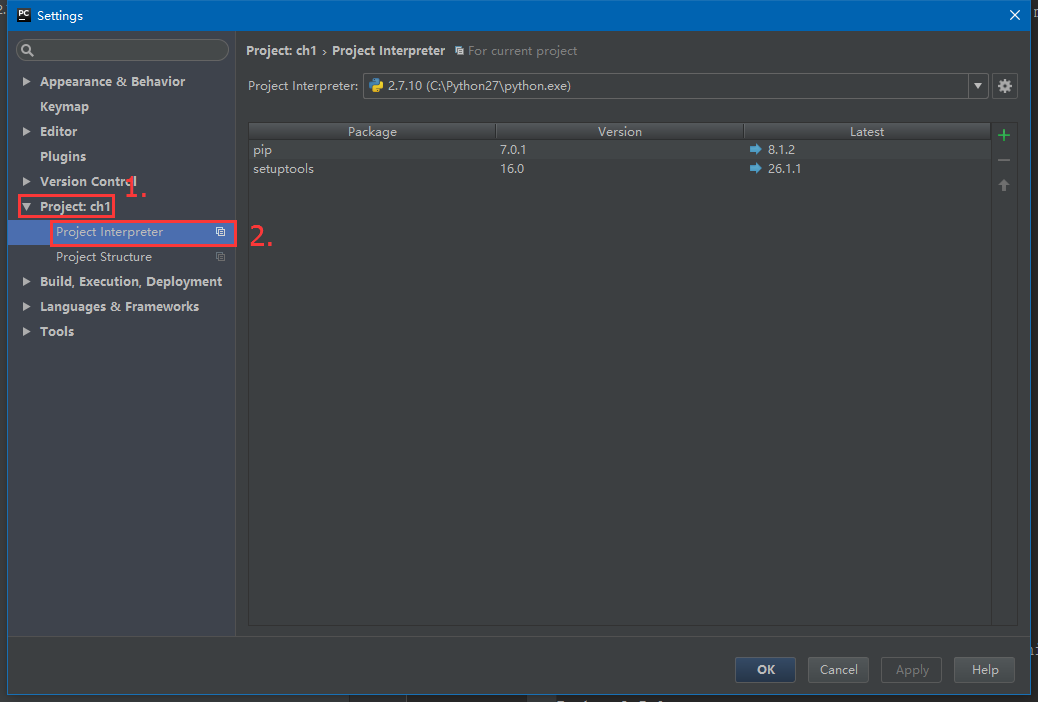
当前项目使用的是 **python2** ：



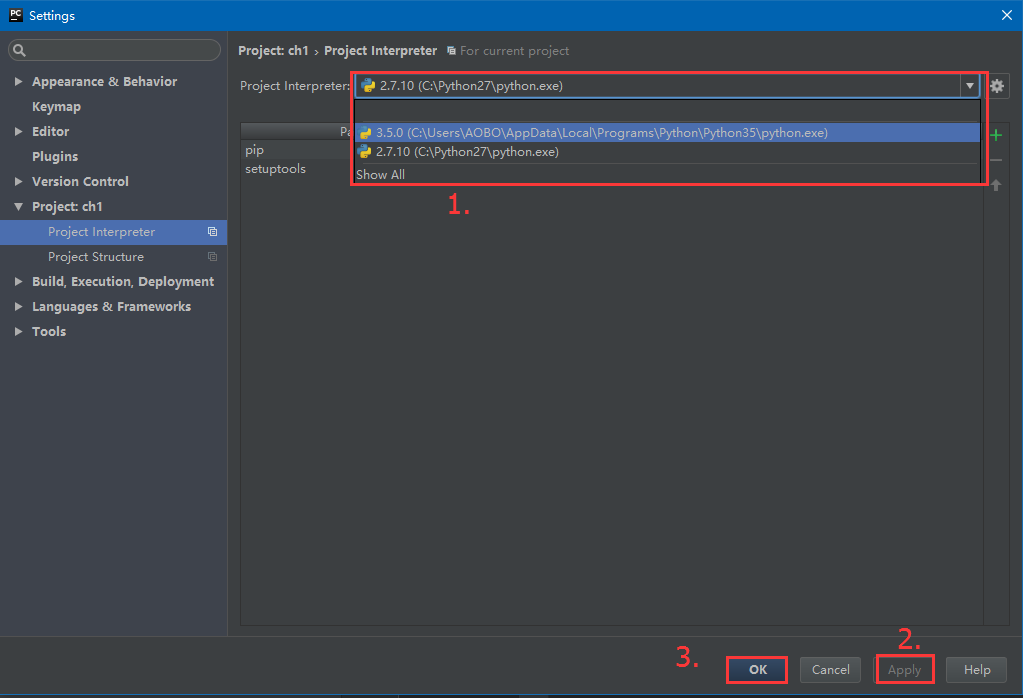
我希望将其设置为 **python3**，操作如下：   
**Step 1 . File** - **Settings**

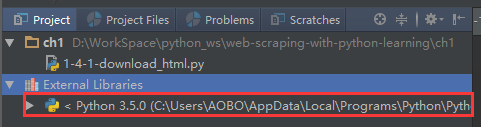


**Step 2 . Project [项目名]** - **Project Interpreter**



**Step 3 . 选择你需要的python版本** - **Apply** - **OK**





## 封装基础

### 一、什么是封装

　　在程序设计中，封装（Encapsulation）是对具体对象的一种抽象，即将某些部分隐藏起来，在程序外部看不到，其

含义是其他程序无法调用。

　　要了解封装，离不开“私有化”，就是将类或者是函数中的某些属性限制在某个区域之内，外部无法调用。

### 二、为什么要封装

　　封装数据的主要原因是：保护隐私（把不想别人知道的东西封装起来）

　　封装方法的主要原因是：隔离复杂度（比如：电视机，我们看见的就是一个黑匣子，其实里面有很多电器元件，对于

用户来说，我们不需要清楚里面都有些元件，电视机把那些电器元件封装在黑匣子里，提供给用户的只是几个按钮接口，

通过按钮就能实现对电视机的操作。）

　　提示：在编程语言里，对外提供的接口（接口可理解为了一个入口），就是函数，称为接口函数，这与接口的概念还

不一样，接口代表一组接口函数的集合体。

### 三、封装分为两个层面

　　封装其实分为两个层面，但无论哪种层面的封装，都要对外界提供好访问你内部隐藏内容的接口（接口可以理解为入

口，有了这个入口，使用者无需且不能够直接访问到内部隐藏的细节，只能走接口，并且我们可以在接口的实现上附加更

多的处理逻辑，从而严格控制使用者的访问）

　　第一个层面的封装（什么都不用做）：创建类和对象会分别创建二者的名称空间，我们只能用类名.或者obj.的方式去

访问里面的名字，这本身就是一种封装。

|  |  |
| --- | --- |
| 1  2  3  4  5 | print(m1.brand) #实例化对象（m1.）  print(motor\_vehicle.tag) #类名（motor\_vehicle.）  -------------输出结果--------------  春风  fuel oil |

　　注意：对于这一层面的封装（隐藏），类名.和实例名.就是访问隐藏属性的接口

第二个层面的封装：类中把某些属性和方法隐藏起来(或者说定义成私有的)，只在类的内部使用、外部无法访问，或

者留下少量接口（函数）供外部访问。

　　Python中私有化的方法也比较简单，即在准备私有化的属性（包括方法、数据）名字前面加两个下划线即可。

　　类中所有双下划线开头的名称如\_\_x都会自动变形成：\_类名\_\_x的形式：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class A:      \_\_N=0 #类的数据属性就应该是共享的,但是语法上是可以把类的数据属性设置成私有的如\_\_N,会变形为\_A\_\_N      def \_\_init\_\_(self):          self.\_\_X=10 #变形为self.\_A\_\_X      def \_\_foo(self): #变形为\_A\_\_foo          print('from A')      def bar(self):          self.\_\_foo() #只有在类内部才可以通过\_\_foo的形式访问到. |

　　这种自动变形的特点：

　　　　1、类中定义的\_\_x只能在内部使用，如self.\_\_x，引用的就是变形的结果。

　　　　2、这种变形其实正是针对外部的变形，在外部是无法通过\_\_x这个名字访问到的。

　　　　3、在子类定义的\_\_x不会覆盖在父类定义的\_\_x，因为子类中变形成了：\_子类名\_\_x,而父类中变形成了：\_父

类名\_\_x，即双下滑线开头的属性在继承给子类时，子类是无法覆盖的。

　　注意：对于这一层面的封装（隐藏），我们需要在类中定义一个函数（接口函数）在它内部访问被隐藏的属性，然后

外部就可以使用了

　　这种变形需要注意的问题是：

　　1、这种机制也并没有真正意义上限制我们从外部直接访问属性，知道了类名和属性名就可以拼出名字：\_类名\_\_属

性，然后就可以访问了，如a.\_A\_\_N

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | a = A()  print(a.\_A\_\_N)  print(a.\_A\_\_X)  print(A.\_A\_\_N)  --------输出结果--------  0  10  0 |

　　2、变形的过程只在类的定义是发生一次,在定义后的赋值操作，不会变形

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = A() #实例化对象a  print(a.\_\_dict\_\_) #打印变形的内容  a.\_\_Y = 20 #新增Y的值，此时加\_\_不会变形  print(a.\_\_dict\_\_) #打印变形的内容  ---------输出结果----------  {'\_A\_\_X': 10}  {'\_A\_\_X': 10, '\_\_Y': 20} #发现后面的Y并没有变形 |

　　3、在继承中，父类如果不想让子类覆盖自己的方法，可以将方法定义为私有的

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class A: #这是正常情况      def fa(self):          print("from A")      def test(self):          self.fa()    class B(A):      def fa(self):          print("from B")    b = B()  b.test()  --------输出结果----------  from B |

　　看一下把fa被定义成私有的情况：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class A: #把fa定义成私有的，即\_\_fa      def \_\_fa(self): #在定义时就变形为\_A\_\_fa          print("from A")      def test(self):          self.\_\_fa() #只会与自己所在的类为准,即调用\_A\_\_fa    class B(A):      def \_\_fa(self): #b调用的是test，跟这个没关系          print("from B")    b = B()  b.test()  -------输出结果---------  from A |

### 四、特性（property）

　　1、什么是特性property

　　property是一种特殊的属性，访问它时会执行一段功能（函数）然后返回值（就是一个装饰器）

　　注意：被property装饰的属性会优先于对象的属性被使用，而被propery装饰的属性,分成三种：property、被装饰

的函数名.setter、被装饰的函数名.deleter（都是以装饰器的形式）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | class room: #定义一个房间的类      def \_\_init\_\_(self,length,width,high):          self.length = length #房间的长          self.width = width #房间的宽          self.high = high #房间的高      @property      def area(self): #求房间的平方的功能          return self.length \* self.width #房间的面积就是：长x宽      @property      def perimeter(self): #求房间的周长的功能          return 2 \* (self.length + self.width) #公式为：（长 + 宽）x 2      @property      def volume(self): #求房间的体积的功能          return self.length \* self.width \* self.high #公式为：长 x 宽 x 高    r1 = room(2,3,4) #实例化一个对象r1  print("r1.area：",r1.area) #可以像访问数据属性一样去访问area,会触发一个函数的执行,动态计算出一个值  print("r1.perimeter：",r1.perimeter) #同上，就不用像调用绑定方法一样，还得加括号，才能运行  print("r1.volume：",r1.volume) #同上，就像是把运算过程封装到一个函数内部，我们不管过程，只要有结果就行  ------------输出结果---------------  r1.area： 6  r1.perimeter： 10  r1.volume： 24 |

 　　注意：此时的特性arear、perimeter和volume不能被赋值。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | r1.area = 8 #为特性area赋值  r1.perimeter = 14 #为特性perimeter赋值  r1.volume = 24 #为特性volume赋值  '''  抛出异常：      r1.area = 8 #第一个就抛异常了，后面的也一样  AttributeError: can't set attribute    ''' |

2、为什么要用property

　　将一个类的函数定义成特性以后，对象再去使用的时候obj.name,根本无法察觉自己的name是执行了一个函数然后

计算出来的，这种特性的使用方式遵循了统一访问的原则。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | class people: #定义一个人的类      def \_\_init\_\_(self,name,sex):          self.name = name          self.sex = sex #p1.sex = "male",遇到property，优先用property        @property #查看sex的值      def sex(self):          return self.\_\_sex #返回正真存值的地方        @sex.setter #修改sex的值      def sex(self,value):          if not isinstance(value,str): #在设定值之前进行类型检查              raise TypeError("性别必须是字符串类型") #不是str类型时，主动抛出异常          self.\_\_sex = value #类型正确的时候，直接修改\_\_sex的值，这是值正真存放的地方              #这里sex前加"\_\_",对sex变形，隐藏。        @sex.deleter #删除sex      def sex(self):          del self.\_\_sex    p1 = people("egon","male") #实例化对象p1  print(p1.sex) #查看p1的sex，此时要注意self.sex的优先级  p1.sex = "female" #修改sex的值  print(p1.sex) #查看修改后p1的sex  print(p1.\_\_dict\_\_) #查看p1的名称空间，此时里面有sex  del p1.sex #删除p1的sex  print(p1.\_\_dict\_\_) #查看p1的名称空间，此时发现里面已经没有sex了  -------------------输出结果--------------------  male  female  {'name': 'egon', '\_people\_\_sex': 'female'}  {'name': 'egon'} |

　　python并没有在语法上把它们三个内建到自己的class机制中，在C++里一般会将所有的所有的数据都设置为私有的

，然后提供set和get方法（接口）去设置和获取，在python中通过property方法可以实现。

### 五、封装与扩展性

　　封装在于明确区分内外，使得类实现者可以修改封装内的东西而不影响外部调用者的代码；而外部使用用者只知道一

个接口(函数)，只要接口（函数）名、参数不变，使用者的代码永远无需改变。这就提供一个良好的合作基础——或者说

，只要接口这个基础约定不变，则代码改变不足为虑。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #类的设计者  class room: #定义一个房间的类      def \_\_init\_\_(self,name,owner,length,width,high):          self.name = name          self.owner = owner          self.\_\_length = length #房间的长          self.\_\_width = width #房间的宽          self.\_\_high = high #房间的高      @property      def area(self): #求房间的平方的功能          return self.\_\_length \* self.\_\_width #对外提供的接口，隐藏了内部的实现细节，\                                              # 此时我们想求的是房间的面积就是：长x宽 |

　　实例化对象通过接口，调用相关属性得到想要的值：

|  |  |
| --- | --- |
| 1  2  3  4  5 | #类的使用者  r1 = room("客厅","michael",20,30,9) #实例化一个对象r1  print(r1.area) #通过接口使用（area），使用者得到了客厅的面积  -------------输出结果--------------  600 #得到了客厅的面积 |

　　扩展原有的代码，使功能增加：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | #类的设计者，轻松的扩展了功能，而类的使用者完全不需要改变自己的代码  class room: #定义一个房间的类      def \_\_init\_\_(self,name,owner,length,width,high):          self.name = name #房间名          self.owner = owner #房子的主人          self.\_\_length = length #房间的长          self.\_\_width = width #房间的宽          self.\_\_high = high #房间的高      @property      def area(self): #对外提供的接口，隐藏内部实现          return self.\_\_length \* self.\_\_width,\                 self.\_\_length \* self.\_\_width \* self.\_\_high #此时我们增加了求体积,          # 内部逻辑变了,只需增加这行代码就能简单实现,而且外部调用感知不到,仍然使          # 用该方法，但是功能已经增加了 |

　　对于类的使用者，仍然在调用area接口的人来说，根本无需改动自己的代码，就可以用上新功能：

|  |  |
| --- | --- |
| 1  2  3  4  5 | #类的使用者  r1 = room("客厅","michael",20,30,9) #实例化一个对象r1  print(r1.area) #通过接口使用（area），使用者得到了客厅的面积  --------------输出结果---------------  (600, 5400) #得到了新增的功能的值 |

## 导入库

Python导入模块的方法有两种：import module 和 from module import，区别是前者所有导入的东西使用时需加上模块名的限定，而后者不要。  
正确的代码：  
>>> import Person  
>>> person = Person.Person('dnawo','man')  
>>> print person.Name  
或  
>>> from Person import \*  
>>> person = Person('dnawo','man')  
>>> print person.Name

# Python2

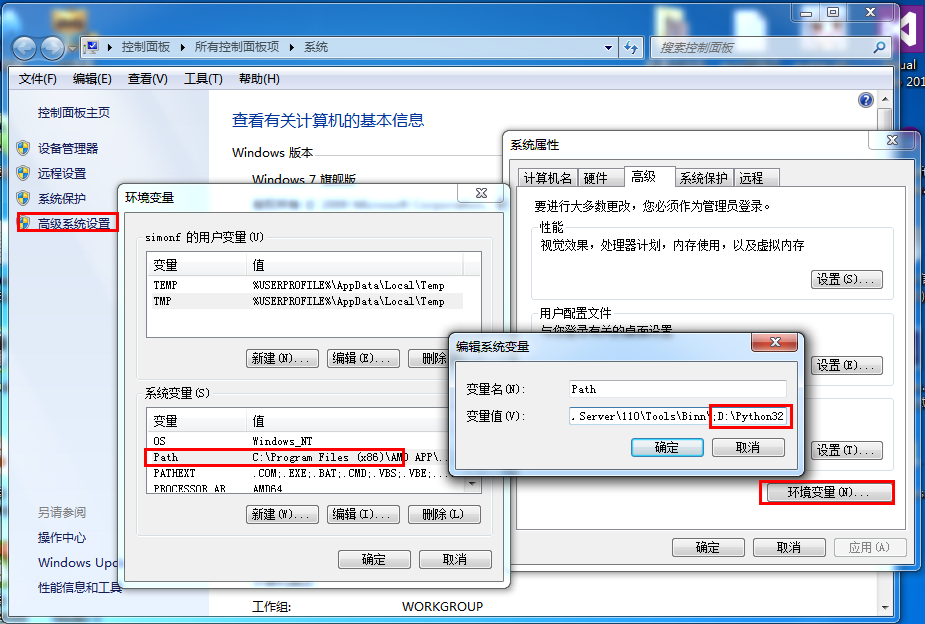
## Windows Python2安装

1、首先访问<http://www.python.org/download/>去下载最新的python版本。

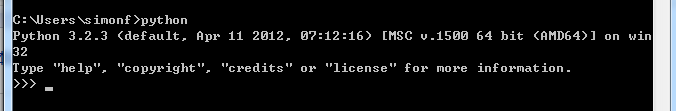


2、安装下载包，一路next。

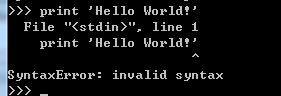
3、为计算机添加安装目录搭到环境变量，如图把python的安装目录添加到pth系统变量中即可。



4、测试python安装是否成功，cmd打开命令行输入 python 命令，如下图即成功了



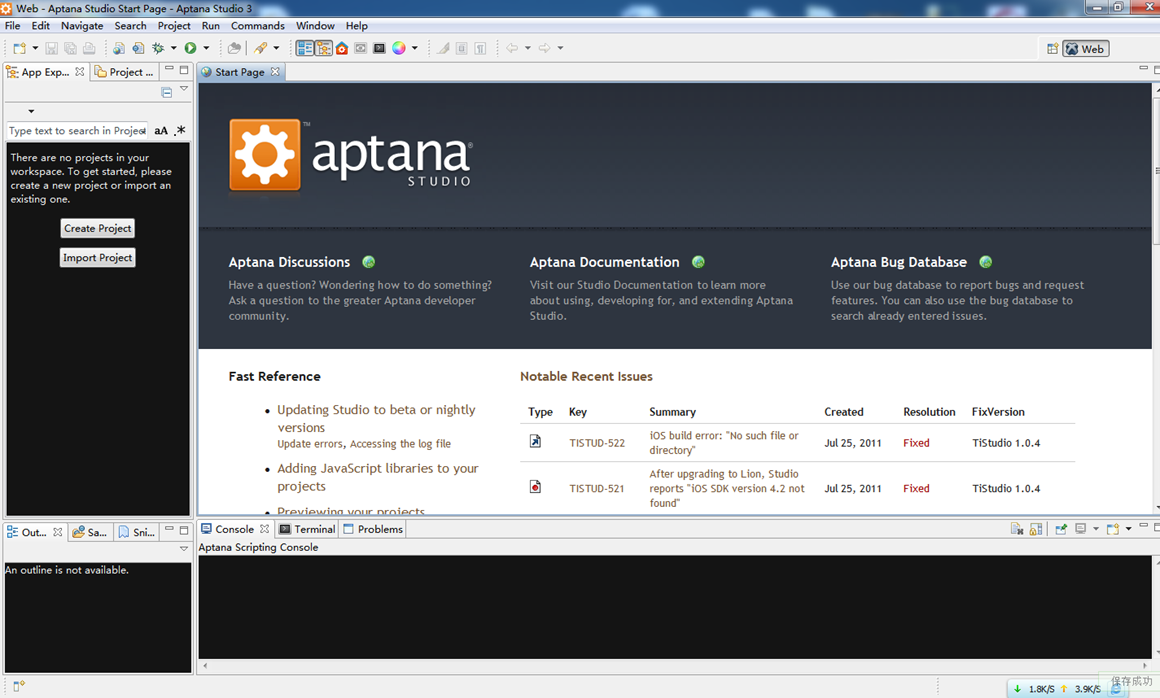
5、Hello World！ 按照很多资料上写的，输入 print 'Hello World!'



居然是不成功，说语法错误！这是一个坑了，很多资料是都是python 1.\*/2.\*的写法，现在我们装的是3.2版本，要采用这种写法 print ('Hello World!')，需要用括号把字符串包含

http://images.cnblogs.com/cnblogs_com/windinsky/201209/201209201708006777.png

6、经过以上测试，python环境算是装好了，但是怎么开发呢，难道用这种命令行？同样有IDE可以应用，请下载Aptana Studio这款IDE工具，这款工具是在Eclipse的基础上加插件集改的，支持python开发。到<http://aptana.com/products/studio3/download>下载安装，然后运行



或者也可以装好Eclipse后自己搜索PyDev插件安装。

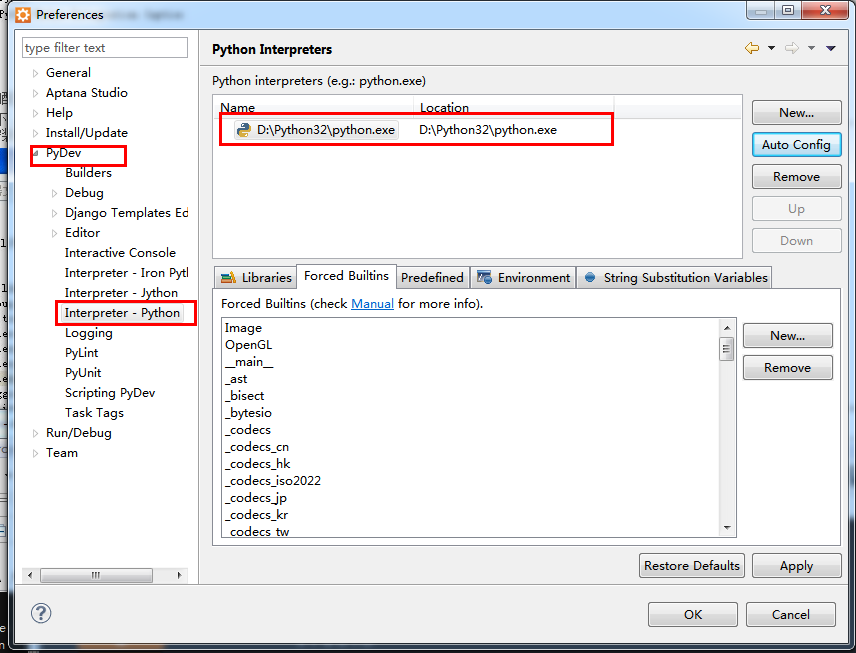
 遇到错误failed to correctly acquire installer\_nodejs\_windows

说明需要代理

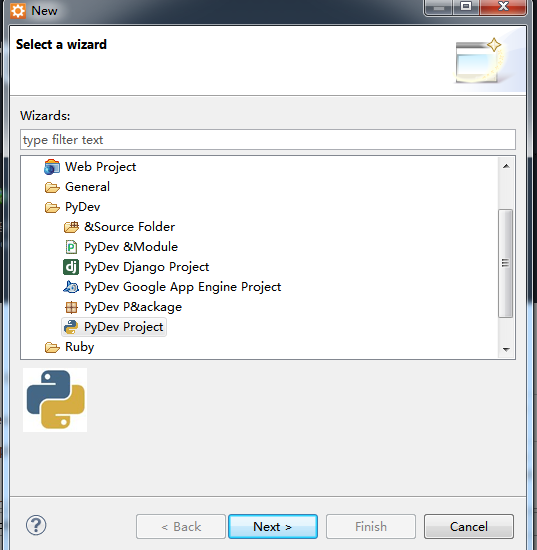
7、ok，在开始之前，也要为这款IDE配置一下环境变量。

打开 窗口->首选项(Window->Preferences...) 对话框，从右侧的树形列表中选择" PyDev"->"Interpreter Python"， 点击新建（New）按钮，从Python的安装路径下选择Python.exe，然后会弹出一个对话框让你勾选System PYTHONPATH，点确定，大功告成。

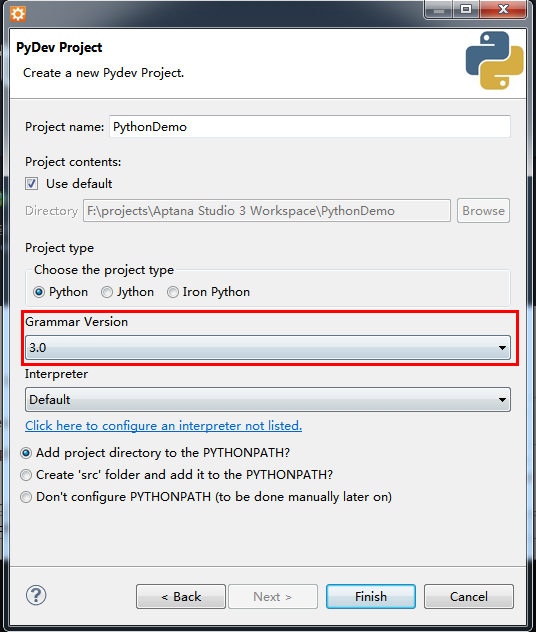
也可以点Auto Config自动配置，据说会搜索安装好的python自动配置，不过我没成功。



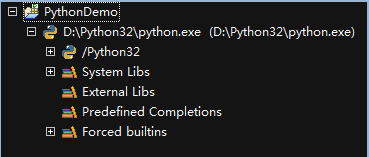
8、新建一个项目



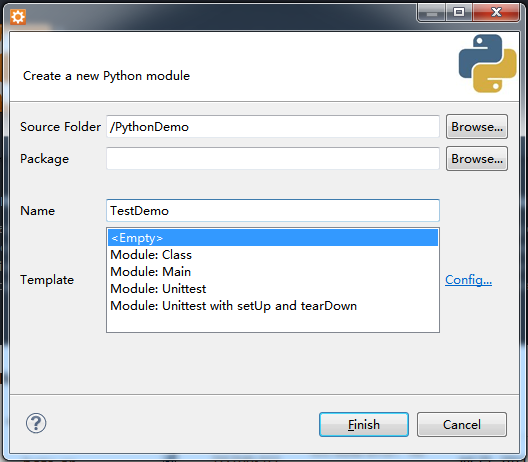
注意一下，因为安装的python是3.2版本，所以这里的语法版本选择3.0（默认2.6）



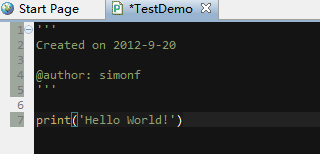
项目建立成功



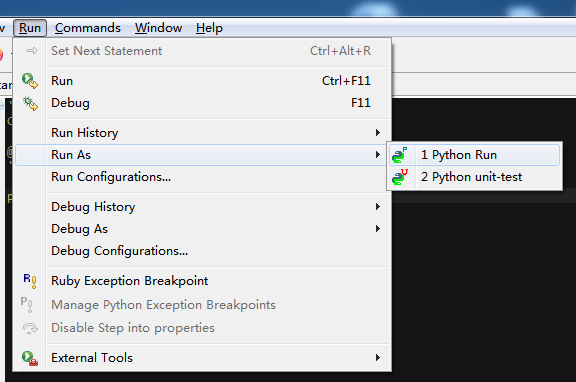
9、在目录下新建一个python文件，输出Hello World！



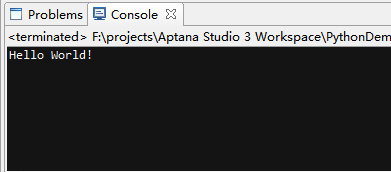
输入代码



开始运行



然后在下方的Console面板中会出现结果



ok，初期工作就是这样了。

## 语法

### 后台执行

### 退出程序

os.\_exit()， sys.exit()

### 判断系统平台

import platform

import sys

import os

def isWindowsSystem():

return 'Windows' in platform.system()

def isLinuxSystem():

return 'Linux' in platform.system()

print isWindowsSystem()

print isLinuxSystem()

print platform.system()

### 遍历目录

Walker

### 字符串

#### 去空格及特殊符号

#### s.strip().lstrip().rstrip(',')

#### 复制字符串

#strcpy(sStr1,sStr2)

sStr1 = 'strcpy'

sStr2 = sStr1

sStr1 = 'strcpy2'

print sStr2

#### 连接字符串

#strcat(sStr1,sStr2)

sStr1 = 'strcat'

sStr2 = 'append'

sStr1 += sStr2

print sStr1

#### 查找字符

#strchr(sStr1,sStr2)

# < 0 为未找到

sStr1 = 'strchr'

sStr2 = 's'

nPos = sStr1.index(sStr2)

print nPos

#### 比较字符串

#strcmp(sStr1,sStr2)

sStr1 = 'strchr'

sStr2 = 'strch'

print cmp(sStr1,sStr2)

#### 扫描字符串是否包含指定的字符

#strspn(sStr1,sStr2)

sStr1 = '12345678'

sStr2 = '456'

#sStr1 and chars both in sStr1 and sStr2

print len(sStr1 and sStr2)

#### 字符串长度

#strlen(sStr1)

sStr1 = 'strlen'

print len(sStr1)

#### 将字符串中的大小写转换

#strlwr(sStr1)

sStr1 = 'JCstrlwr'

sStr1 = sStr1.upper()

#sStr1 = sStr1.lower()

print sStr1

#### 追加指定长度的字符串

#strncat(sStr1,sStr2,n)

sStr1 = '12345'

sStr2 = 'abcdef'

n = 3

sStr1 += sStr2[0:n]

print sStr1

#### 字符串指定长度比较

#strncmp(sStr1,sStr2,n)

sStr1 = '12345'

sStr2 = '123bc'

n = 3

print cmp(sStr1[0:n],sStr2[0:n])

#### 复制指定长度的字符

#strncpy(sStr1,sStr2,n)

sStr1 = ''

sStr2 = '12345'

n = 3

sStr1 = sStr2[0:n]

print sStr1

#### 将字符串前n个字符替换为指定的字符

#strnset(sStr1,ch,n)

sStr1 = '12345'

ch = 'r'

n = 3

sStr1 = n \* ch + sStr1[3:]

print sStr1

#### 扫描字符串

#strpbrk(sStr1,sStr2)

sStr1 = 'cekjgdklab'

sStr2 = 'gka'

nPos = -1

for c in sStr1:

if c in sStr2:

nPos = sStr1.index(c)

break

print nPos

#### 翻转字符串

#strrev(sStr1)

sStr1 = 'abcdefg'

sStr1 = sStr1[::-1]

print sStr1

#### 查找字符串

#strstr(sStr1,sStr2)

sStr1 = 'abcdefg'

sStr2 = 'cde'

print sStr1.find(sStr2)

#### 分割字符串

#strtok(sStr1,sStr2)

sStr1 = 'ab,cde,fgh,ijk'

sStr2 = ','

sStr1 = sStr1[sStr1.find(sStr2) + 1:]

print sStr1

#或者

s = 'ab,cde,fgh,ijk'

print(s.split(','))

#### 连接字符串

delimiter = ','

mylist = ['Brazil', 'Russia', 'India', 'China']

print delimiter.join(mylist)

#### PHP 中 addslashes 的实现

def addslashes(s):

d = {'"':'\\"', "'":"\\'", "\0":"\\\0", "\\":"\\\\"}

return ''.join(d.get(c, c) for c in s)

s = "John 'Johny' Doe (a.k.a. \"Super Joe\")\\\0"

print s

print addslashes(s)

#### 只显示字母与数字

def OnlyCharNum(s,oth=''):

s2 = s.lower();

fomart = 'abcdefghijklmnopqrstuvwxyz0123456789'

for c in s2:

if not c in fomart:

s = s.replace(c,'');

return s;

print(OnlyStr("a000 aa-b"))

#### 截取字符串

str = ’0123456789′  
print str[0:3] #截取第一位到第三位的字符  
print str[:] #截取字符串的全部字符  
print str[6:] #截取第七个字符到结尾  
print str[:-3] #截取从头开始到倒数第三个字符之前  
print str[2] #截取第三个字符  
print str[-1] #截取倒数第一个字符  
print str[::-1] #创造一个与原字符串顺序相反的字符串  
print str[-3:-1] #截取倒数第三位与倒数第一位之前的字符  
print str[-3:] #截取倒数第三位到结尾  
print str[:-5:-3] #逆序截取，具体啥意思没搞明白？

#### 获取长度

len(x)

#### 访问

x[i] 从0开始

## 错误

### IndentationError: expected an indented block

缩进问题

### UnicodeDecodeError: 'ascii' codec can't decode byte 0xe5 in position 42: ordinal not in range(128)

．原因分析：字符问题。在Windows系统转Linux系统时，字符问题很容易出现。

3．解决办法：在出现问题的页加上如下三行即可：

import sys

reload(sys)

sys.setdefaultencoding('utf-8')

## Pexpect

### 安装

先安装python3

再安装ptyprocess

**python setup.py build && python setup.py install**

最后安装pexpect即可

测试:

import pexpect

>>> dir(pexpect)

#### 资料

pexpect是python交互模块，有两种使用方法，一种是函数：run另外一种是spawn类

1.pexpect  module 安装

tar -xzvf pexpect-3.3.tar.gz

cd pexpect-3.3

python setup install (require root)

但是 这个安装需要root权限，如果没有root权限在，还能使用吗？

答案是肯定的，你只需要把lib的路径放入sys.path。这样便可以使用pexpect

#!/usr/bin/env python

import sys

sys.path.append('pexpect-3.3/build/lib')

　　确认安装成功：

>>> import pexpect

>>> dir(pexpect)

['EOF', 'ExceptionPexpect', 'PY3', 'TIMEOUT', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_path\_\_', '\_\_revision\_\_', '\_\_version\_\_', '\_run', 'codecs', 'errno', 'fcntl', 'is\_executable\_file', 'os', 'pty', 're', 'resource', 'run', 'runu', 'searcher\_re', 'searcher\_string', 'select', 'signal', 'spawn', 'spawnu', 'split\_command\_line', 'stat', 'struct', 'sys', 'termios', 'time', 'traceback', 'tty', 'types', 'which']

[复制代码](javascript:void(0);)

2.使用方法

　　run 函数,run函数和os。system()差不多，所不同的是os.system返回的是整数，而run返回字符串

[复制代码](javascript:void(0);)

>>> print pexpect.run('ping localhost -c 3')

PING localhost (127.0.0.1) 56(84) bytes of data.

64 bytes from localhost (127.0.0.1): icmp\_seq=1 ttl=64 time=0.087 ms

64 bytes from localhost (127.0.0.1): icmp\_seq=2 ttl=64 time=0.088 ms

64 bytes from localhost (127.0.0.1): icmp\_seq=3 ttl=64 time=0.088 ms

--- localhost ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2002ms

rtt min/avg/max/mdev = 0.087/0.087/0.088/0.010 ms

[复制代码](javascript:void(0);)

 spawn 类是通过生成子程序sendline发送命令与expect拿回返回进行交互

import pexpect  
child = pexpect.spawn('python',timeout=3)  
child.expect(">>>")  
child.sendline("exit()")  
print child.before   # Print the result of the ls command.

timeout是等待时间，如果超过就会抛出exception，可以 使用except关键字捕获

设置log

fout=file('filename','a') #w write /a append  
child = pexpect.spawn('su root')

child.logfile = sys.stdout  
child.logfile\_send=fout

[复制代码](javascript:void(0);)

#!/usr/bin/python

'''

this script can batch add user

everytime will add specific usename

user mumber and password

create by Young 2014/08/02

require pexpect module, if you don't have one

please install this module

if you can't install this module please

add this module path to sys.path

'''

import pexpect

import getopt

import os

import sys

import random

import string

# usage fuction

def usage():

print '''

Usage: python %s --name user --amount 100 --password [optional]

or python %s -n user -a 100 -p [optional]

this will create user1~user100 and default password will be random or specific.

make sure when you run this script as root

-n,--name the username you want create

-a,--amount the amount of users you want create

-p,--password the default password of use your create

-h,--help display this help and exit

-v,--version output version information and exit

''' %(sys.argv[0],sys.argv[0])

# get the parameters

# set the default user name

user\_name='user'

# generate random password

def set\_password():

word=[x for x in 'aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ0123456789']

p=string.join(random.sample(word, 8)).replace(" ","")

return p

# number of users ,set default users number as 10

number\_of\_users=10

password=set\_password()

is\_ramdom\_password=True

#print password

def get\_command():

try:

opts, args = getopt.getopt(sys.argv[1:], "n:a:p:hv", ["name=","amount=","password=","help", "version"])

except getopt.GetoptError as err:

# print help information and exit

print str(err) # will print something like "option -a not recognized"

usage()

sys.exit(2)

#print opts,args

for option,value in opts:

#print "----------->"+option+' <><>'+value

if option in ["-n","--name"]:

if(len(value)>=4):

global user\_name

user\_name=value

else:

print "invaild usename,will use user"

elif option in ["-a","--amount"]:

if value.isdigit():

if( int(str(value))>0 and int(str(value))< 1000 ):

global number\_of\_users

number\_of\_users=int(str(value))

else:

usage()

sys.exit(2)

else:

print "ValueError: invalid literal for ",value

print "invaild amount,will use defualt"

elif option in ["-p","--password"]:

if(len(value)>=6):

global password

password=value

global is\_ramdom\_password

is\_ramdom\_password=False

else:

print "invaild password,will use random"

elif option in ["-v","--version"]:

print sys.argv[0]+' 1.0.0'

sys.exit(0)

elif option in ("-h", "--help"):

usage()

sys.exit(0)

else :

assert False, "unhandled option"

usage()

sys.exit(2)

def check\_root():

if( os.environ['USER']!='root'):

print 'Permission denied,please su root'

sys.exit()

#use pexpect to adduser

def run\_add(user,mypassword):

log = file('adduser.log','a')

flag=os.system('adduser '+user)

if(flag!=0):

os.system('userdel '+user)

os.system('adduser '+user)

try:

child=pexpect.spawn('passwd '+user,timeout=5)

child.logfile = log

child.logfile\_send=sys.stdout

child.expect("New password:")

child.sendline(mypassword)

child.expect("Retype new password:")

child.sendline(mypassword)

child.expect("passwd: all authentication tokens updated successfully.")

except pexpect.EOF:

pass

except pexpect.TIMEOUT:

pass

def add\_user(name,amount,password,is\_ramdom\_password):

check\_root()

for number in range(1,amount+1):

if(is\_ramdom\_password):

print "%4d: adduser %s%-4d password %s " %(number,name,number,set\_password())

run\_add(name+str(number),set\_password())

else:

print "%4d: adduser %s%-4d password %s " %(number,name,number,password)

run\_add(name+str(number),set\_password)

get\_command()

add\_user(user\_name,number\_of\_users,password,is\_ramdom\_password)

[复制代码](javascript:void(0);)

### 错误

#### ImportError: No module named 'ptyprocess'

安装ptyprocess

#### [ImportError: No module named pxssh](http://stackoverflow.com/questions/21883912/importerror-no-module-named-pxssh)

from pexpect import pxssh

## Paramiko

与python3不兼容,用2

### 1、简介

　　大家会发现，常见的解决方法都会需要对远程服务器必要的配置，如果远程服务器只有一两台还好说，如果有N台，还需要逐台进行配置，或者需要使用代码进行以上操作时，上面的办法就不太方便了。

　　使用paramiko可以很好的解决以上问题，比起前面的方法，它仅需要在本地上安装相应的软件（python以及PyCrypto），对远程服务器没有配置要求，对于连接多台服务器，进行复杂的连接操作特别有帮助。

### 2、安装

　　安装paramiko有两个先决条件，python和另外一个名为PyCrypto的模块。

　　通常安装标准的python模块，只需要在模块的根目录下运行：

python setup.py build

python setup.py install

备注：安装前先检查是否安装gcc（yum -y install gcc）

#### 2.1 PyCrypto安装

wget http://ftp.dlitz.net/pub/dlitz/crypto/pycrypto/pycrypto-2.6.tar.gz

tar -zxvf pycrypto-2.6.tar.gz

cd pycrypto-2.6/

python setup.py build && python setup.py install

　　测试：

python>> import Crypto

(编译时报错：error: command 'gcc' failed with exit status 1；这是因为缺少python-dev的软件包，所yum -y install python-devel)

#### 2.2 paramiko安装

wget http://www.lag.net/paramiko/download/paramiko-1.7.7.1.tar.gz

tar xvzf paramiko-1.7.7.1.tar.gz

cd paramiko-1.7.7.1/

python setup.py build && python setup.py install

Crypto error: 'module' object has no attribute 'HAVE\_DECL\_MPZ\_POWM\_SEC'

　　测试：

python>> import paramiko

　　(Crypto error: 'module' object has no attribute 'HAVE\_DECL\_MPZ\_POWM\_SEC'

　　找到 /usr/lib/python2.7/site-packages/Crypto/Util/number.py

　　把if \_fastmath is not None and not \_fastmath.HAVE\_DECL\_MPZ\_POWM\_SEC:

　　注释了

　　#if \_fastmath is not None and not \_fastmath.HAVE\_DECL\_MPZ\_POWM\_SEC:

)

### 错误

#### ImportError: cannot import name OSRNG

安装ecdsa

### 3、使用

#### 3.1 执行远程命令

#!/usr/bin/python

import paramiko

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect("某IP地址",22,"用户名", "口令")

stdin, stdout, stderr = ssh.exec\_command("你的命令")

print stdout.readlines()

ssh.close()

#### 3.2 上传文件到远程

#!/usr/bin/python

import paramiko

t = paramiko.Transport(("某IP地址",22))

t.connect(username = "用户名", password = "口令")

sftp = paramiko.SFTPClient.from\_transport(t)

remotepath='/tmp/test.txt'

localpath='/tmp/test.txt'

sftp.put(localpath,remotepath)

t.close()

#### 3.3 从远程下载文件

#!/usr/bin/python

import paramiko

t = paramiko.Transport(("某IP地址",22))

t.connect(username = "用户名", password = "口令")

sftp = paramiko.SFTPClient.from\_transport(t)

remotepath='/tmp/test.txt'

localpath='/tmp/test.txt'

sftp.get(remotepath, localpath)

t.close()

# pyqt4

## 载入qdesinger设计的ui组件

import sys

from PyQt4 import QtCore, QtGui, uic

**qtCreatorFile = "C:\\Python27\\Lib\\site-packages\\PyQt4\\test.ui"**

#load qtui

Ui\_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MyApp(QtGui.QMainWindow, Ui\_MainWindow):

def \_\_init\_\_( self ):

QtGui.QMainWindow.\_\_init\_\_( self )

Ui\_MainWindow.\_\_init\_\_( self )

self .setupUi( self )

if \_\_name\_\_ == "\_\_main\_\_" :

app = QtGui.QApplication(sys.argv)

window = MyApp()

window.show()

sys.exit(app.exec\_())

# eclipse python 开发环境

在Eclipse中安pydev插件

启动Eclipse->点击Help->Install New Software...->弹出的对话框中点Add

->Name填:Pydev   -> Location填http://pydev.org/updates ->然后OK就行

安装完成后，在 Eclipse 菜单栏中，选择Window > Preferences > Pydev > Python Interpreter>New ，继续配置解释器

Quick autoconfig