python分布式爬虫打造搜索引擎

作者网址：<http://projectsedu.com/>

# 第一章 课程介绍

## 课程介绍

# 第二章 环境搭建

## pycharm的安装和使用

### pycharm在windows下的安装

pycharm的下载地址：

<http://www.jetbrains.com/pycharm/download/#section=windows>

下载的时候请下载Professional专业版本，注册码地址：<http://idea.lanyus.com/>

安装很简单，跟其他软件的安装方式差不多。

### pycharm在linux下的安装

下载地址：

<http://www.jetbrains.com/pycharm/download/#section=linux>

下载的时候请下载Professional专业版本，注册码地址：<http://idea.lanyus.com/>

下载的是一个压缩包，在Linux中把压缩包下载下来之后，直接进行解压缩。

解压缩完成之后，进入 pycharm/bin 目录，在该目录下的 pycharm.sh 就是用来启动 pycharm 的，输入 ./pycharm.sh 启动 pycharm 编辑器。

#### pycharm.sh 的快捷启动

输入 vim ~/.bashrc ，编辑用户目录下的 .bashrc 文件

在文件里面添加一行

alias pycharm=”bash /home/hobby/pycharm/bin/pycharm.sh”



然后输入 source ~/.bashrc 使该配置文件生效。

以后直接输入 pycharm 就可以启动IDE了。

附带：Linux从任意目录进入当前用户目录的命令 cd ~ 即可。

### pycharm简单操作

**快捷键设置**：启动 pycharm ，File → Settings → 搜索“Keymap”选择自己熟悉的快捷键方式。

**导入工程**：File → Open → 选择工程文件目录即可。

**Python解析器设置**：File → Settings → 搜索“Interpreter”，找到 Project Interpreter，通过 Add Local 把虚拟环境添加进来。

## mysql和navicat的安装和使用

### mysql的下载和安装

**Windows环境**：

打开<https://dev.mysql.com/downloads/windows/>，找到[**MySQL Installer**](https://dev.mysql.com/downloads/installer/)，选择(mysql-installer-community-5.7.18.1.msi)，进行下载和安装。

**Linux环境**：

输入 cd ~进入当前用户目录

然后输入 sudo apt-get install mysql-server

然后输入 ps aux|grep mysqld 可以查看 mysql 是否启动成功

输入 mysql -u用户名 -p 进入mysql

### 配置mysql使主机能访问虚拟机里面的mysql服务器

#### 修改mysql配置文件

输入 sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf

编辑 bind-address 的值，修改为 0.0.0.0 然后保存退出。

然后 sudo service mysql restart 重启mysql服务器。

#### 主机navicat访问虚拟机里mysql

输入 ifconfig 查看虚拟机的IP地址，然后在 navicat中填写虚拟主机的ip地址、mysql用户名和密码，就可以访问了。

#### mysql不允许访问怎么办？

navicat访问mysql出现如下错误怎么处理？

|  |
| --- |
| 1130 – Host ‘IP’ is not allowed to connect to this MySQL server |

这是因为mysql还没有授权的问题，进入 mysql环境，再输入如下命令即可。

|  |
| --- |
| mysql -u用户名 -p密码  mysql>grant all privileges on \*.\* to "用户名"@"%" identified by "密码" with grant option;  mysql>flush privileges; |

就解决了。

说明："用户名"@"%" 中的 % 是指所有的外部IP地址。

### navicat 的安装和使用

可以直接从网上找绿色破解版使用即可。

在使用 navicat创建数据库的时候注意如下两点：

字符集： utf8 -- UTF-8 Unicode

排序规则：utf8\_general\_ci

## python2.7 和 3.5版本的安装和使用

python的下载地址：

<https://www.python.org/downloads/>

### windows环境下

在 windows 环境下安装 python2.7 和 3.5 版本的时候，只有其中一个版本的路径可以加入path环境变量。

使用Python的时候可以先进入Python的安装目录，然后按住Shift+右键，然后打开“在此处打开命令窗口”来使用。

### Linux环境下

sudo apt-get install python3.5

2.7版本是Linux自带的版本，在使用Python的时候，默认使用2.7版本。如果要使用3.5版本，则输入 python3 即可。

## 虚拟环境组件的安装和配置

在安装 Python 组件的时候建议使用国内的镜像文件，比如 豆瓣源

<https://pypi.doubanio.com/simple/>

Python一些安装包安装失败的时候可以从下面网址进行下载

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

### Windows安装Python虚拟环境

#### 安装 virtualenv 环境

pip install virtualenv 安装 virtualenv组件

使用国内镜像安装 virtualenv 环境

pip install -i <https://pypi.doubanio.com/simple/> virtualenv

pip install -i <https://pypi.doubanio.com/simple/> django

pip uninstall django

virtualenv的使用：

virtualenv scrapytest 则会在当前目录下创建名为 scrapytest的虚拟环境

cd ../scrapytest/Scripts 进入scrapytest虚拟环境的Scripts目录下

运行 activate.bat 激活scrapytest虚拟环境

运行 deactivate.bat 退出 scrapytest虚拟环境

指定 Python 版本创建虚拟运行环境

virtualenv -p C:\Python3.5\python.exe scrapypy3

使用 virtualenv创建虚拟环境最不好的地方就是需要记住虚拟环境的安装路径。

#### 安装virtualenvwrapper 环境

pip install virtualenvwrapper

配置virtualenvwrapper 创建的虚拟环境的目录，需要通过电脑系统属性配置环境变量中的系统变量：

变量名 WORKON\_HOME

变量值 想要创建的目录的路径

mkvirtualenv py2scrapy 在WORKON\_HOME目录新建 py2scrapy 虚拟环境

workon 列出WORKON\_HOME目录下所有的虚拟环境

workon 虚拟环境名称 激活指定的虚拟环境

deactivate 退出虚拟环境

指定 Python 版本创建虚拟运行环境

mkvirtualenv --python=C:\Python3.5\python.exe py3scrapy

### Linux安装Python虚拟环境

#### 安装 virtualenv 环境

sudo apt-get install python-virtualenv

virtualenv py2 在当前目录下创建py2的虚拟创建

cd py2/bin 进入 py2 虚拟环境目录

source activate 激活 py2 虚拟环境

deactivate 退出 py2 虚拟环境

指定 Python 版本创建虚拟运行环境

virtualenv -p /usr/bin/python3 py3

使用 virtualenv创建虚拟环境最不好的地方就是需要记住虚拟环境的安装路径。

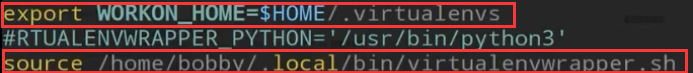
#### 安装virtualenvwrapper 环境

pip install virtualenvwrapper

sudo find / -name virtualenvwrapper.sh

vim ~/.bashrc

添加如图所示红框部分：



$HOME 是指当前用户主目录，修改好之后进行保存。

source ~/.bashrc

使配置生效。

mkvirtualenv py2scrapy

deactivate

workon

workon 虚拟环境名称

指定 Python 版本创建虚拟运行环境

mkvirtualenv --python=/usr/bin/python3 py3scrapy

# 第三章 爬虫基础

## 基础知识

### 技术选型

scrapy or requests beautifulsoup

1、requests和beautifulsoup都是库文件，而scrapy是框架，框架可以集成库文件

2、scrapy框架中可以加入 requests和beautifulsoup库文件

3、scarpy基于twisted（异步IO框架），性能是最大的优势

4、scrapy方便扩展，提供很多内置的功能

5、scrapy内置css和xpath两种html选择器（选择器用c写的，性能高），使用方便，beautifulsoup（用python写的）最大的缺点就是慢

所以，本课程主要使用scrapy和requests，而不使用beautifulsoup。

### 爬虫作用

1. 搜索引擎：百度，谷歌，垂直领域搜索引擎
2. 推荐引擎：今日头条
3. 机器学习的数据样本
4. 数据分析（如：金融数据分析）、舆情分析等……
5. ……

## 正则表达式 Regex

虽然css和xpath选择器可以定位html的元素，但是元素里面的内容有些需要过滤的，就必须依靠正则表达式来完成了，所以正则表达式是需要学习的。

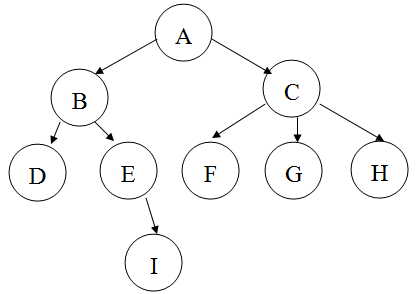
### 特殊字符及使用说明

|  |
| --- |
| ^ $ \* ? + {2} {2,} {2,5} | |
| import re # 正则表达式包  # ^ 表示以 ^ 后面的字符开头  # . 表示任意字符  # \* 表示 \* 前面的字符可以重复任意多次,即 ≥ 0 次  # 如果要控制重复次数的范围,那么 \* 就无法满足,这时可以用下面的方式  # + 表示前面的字符可以重复但至少 1 次  # {2} 表示前面的字符可以重复 2 次  # {2,} 表示前面的字符可以重复 至少 2 次  # {2,5} 表示前面的字符可以重复 2 到 5 次  # $ 表示以 $ 前面的字符结束  # ^zh.\* 表示以 zh 为开头,后面跟上任意字符,字符可以重复任意多次  line = "zhlsh123"  regex\_str = "^zh.\*23$"  if re.match(regex\_str,line):  print("yes")  else:  print("no")  # ? 表示使用非贪婪匹配模式,如果不使用 ? 那么默认使用贪婪匹配模式  # 贪婪匹配模式就是匹配最大长度,或者说是反向匹配,即从最右边开始匹配  # 非贪婪匹配模式则是从最左边开始匹配,并且对 ? 后面的字符或者子串只提取一次  # ( ) 表示提取匹配出来的子串并返回  line = "zhhhhhhhhzzlllshhhh123"  regex\_str = ".\*(z.\*z).\*" # 贪婪匹配模式  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1))  regex\_str = ".\*?(z.\*?z).\*" # 非贪婪匹配模式  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1))  # | 表示 或 的关系,表示 | 两边出现任何一个满足即可  line = "zhhlsh123"  regex\_str = "((zhlsh|zhhlsh)123)" # 非贪婪匹配模式  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1)) # group(1) 表示最外面( )匹配的子串  print(match\_obj.group(2)) # group(2) 表示第二层( )匹配的子串 |
| [ ] [^] [a-z] . |
| import re # 正则表达式包  # [ ] 表示 [ ] 中的字符任意一个满足匹配即可  # [0-9] 表示 [ ] 中的 0~9 中的任意一个数字满足匹配即可  # [^] 表示 [ ] 中不包含 ^ 后面的字符就满足匹配  # [ ] 中包含 . 或者 \* ,则 . 或者 \* 不再有特殊含义,只代表本义  line = "zhhlsh123"  regex\_str = "([xyz]hhlsh123)" # 表示第一个字符为 xyz 中的任意一个满足匹配就匹配成功  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1)) |
| \s \S \w \W |
| # \s 表示空格  # \S 表示非空格  # \w 相当于 [A-Za-z0-9\_] 之内的任意字符  # \W 相当于 [A-Za-z0-9\_] 之外的任意字符  line = "你 好"  regex\_str = "(你\s好)"  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1)) |
| [\u4E00-\u9FA5] ( ) \d |
| import re # 正则表达式包  # [\u4E00-\u9FA5] Unicode编码的汉字,汉字区间  line = "study in 南京大学"  regex\_str = ".\*?([\u4E00-\u9FA5]+大学)"  match\_obj = re.match(regex\_str,line)  if match\_obj:  print(match\_obj.group(1))  # \d 表示数字,即 0~9 中的任意一个数字  # 正则综合应用  line\_1\_1 = "XXX出生于2001年06月01日"  line\_1\_2 = "XXX出生于2001年6月1日"  line\_2\_1 = "XXX出生于2001年06月"  line\_2\_2 = "XXX出生于2001年6月"  line\_3\_1 = "XXX出生于2001/06/01"  line\_3\_2 = "XXX出生于2001/6/1"  line\_4\_1 = "XXX出生于2001-06-01"  line\_4\_2 = "XXX出生于2001-6-1"  line\_5\_1 = "XXX出生于2001-06"  line\_5\_2 = "XXX出生于2001-6"  regex\_str = ".\*(\d{4}[年/-]\d{1,2}([月/-]\d{1,2}(日|$)|[月/-]$|$))"  for i in range(1,6):  for j in range(1,3):  new\_line = 'line\_' + str(i) + '\_' + str(j)  print(new\_line)  match\_obj = re.match(regex\_str,locals()[new\_line])  if match\_obj:  print(match\_obj.group(1))  print("\n") |

## 深度优先和广度优先

### 网站树结构

### 深度优先算法和实现



深度优先的爬取顺序如下：

爬取A，发现A下面有B和C，爬取A的第一条子链接B

爬取B，发现B下面有D和E，爬取B的第一条子链接D

爬取D，发现D下面没有子链接，返回B，爬取第二条子链接E

爬取E，发现E下面有I，爬取E的第一条子链接I，没有其他子链接

到这里B的子链接爬完，返回A，爬取A的第二条子链接

……

按照这样的顺序爬完整个链接结构。

爬取顺序如下：（递归实现）

A→B→D→E→I→C→F→G→H

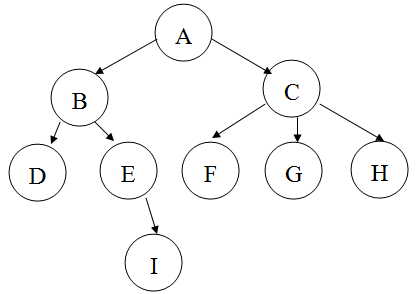
Scrapy默认按照深度优先算法爬取网站链接。

深度优先代码：

|  |
| --- |
| def depth\_tree (tree\_node):  if tree\_node is not None:  print(tree\_node.\_data)  if tree\_node.\_left is not None:  return depth\_tree(tree\_node.\_left)  if tree\_node.\_right is not None:  return depth\_tree(tree\_node.\_right) |

不停的做递归而没有跳出，或者递归调用太深的话，容易造成栈溢出问题。

### 广度优先算法和实现



广度优先的爬取顺序如下：

爬取A，发现A下面有B和C，爬取A的所有子链接B和C

爬取B，发现B下面有D和E，爬取B的所有子链接D和E

爬取C，发现C下面有F、G和H，爬取C的所有子链接F、G和H

爬取D，发现D下面没有子链接

爬取E，发现E下面有I，爬取E的所有子链接I

……

按照这样的顺序爬完整个链接结构。

爬取顺序如下：（队列实现）

A→B→C→D→E→F→G→H→I

深度优先代码：

|  |
| --- |
| def level\_queue (root):  """利用队列实现树的广度优先遍历"""  if root is None:  return  my\_queue = []  node = root  my\_queue.append(node)  while my\_queue:  node = my\_queue.pop(0)  print(node.element)  if node.lchild is not None:  my\_queue.append(node.lchild)  if node.rchild is not None:  my\_queue.append(node.rchild) |

## 爬虫去重策略

**策略一**、将爬取过的url保存到数据库中，当爬取下一个url的时候，就从数据库中查询该url是否已经爬取过，如果有则忽略，如果没有则把该url保存到数据库中。效率非常低。

**策略二**、把访问过的url保存到内存的一个set集合变量中，取url很快，基本上不需要做查询，因此判断url重复就非常快，但是占用内存会越来越大，比如抓取 100000000（一亿）条url地址，占用内存为

100000000\*2byte\*50个字符/1024/1024/1024=9G

**说明**：

2byte是因为Python在内存中是采用Unicode方式的编码，Unicode为固定长度假设为 16个bit，即2个byte

50字符，是指url长度的保守估计的值

因此一条url需要占用2个byte\*50=100byte。

**策略三**、把访问过的url经过md5等方法进行哈希后保存到内存的一个set集合变量中，因为经过md5编码，就可以把url字符转换成固定长度的md5的值，比如 md5编码的值一般为 128个bit即大概是 16byte。相比**策略二**能减少几倍的内存消耗。**Scrapy采用该类似的策略**。

**策略四**、用 bitmap 方法，把访问过的url通过hash函数映射到某一位

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

申请8个bit，即一个byte，每个bit对应一个url。也就是把url计算它的hash函数，然后hash函数把url映射到每一个bit，通过每个bit的0和1来判断该url是否存在。缺点是url通过hash函数映射到bit的冲突非常高，或者说hash函数可能会把多个url映射到同一个bit上。

100000000/8/1024/1024 = 11.9M的内存

**策略五**、通过 bloomfilter 方法对 bitmap 方法进行改进，通过多个hash函数来实现降低冲突。**在Scrapy的分布式爬虫中会用到bloomfilter 方法**。

## 字符串编码

计算机只能处理数字0和1，文本要转换为数字才能处理，在计算机中8个bit作为一个字节byte，所以一个字节能表示最大的数字就是255。

计算机是美国人发明的，所以一个字节可以表示所有的字符了，所以ASCII（一个字节）编码就成为了美国人的标准编码。

处理中文，ASCII编码是不够的，所以中国制定GB2312编码，也就是用2个字节来表示一个汉字，同时把ASCII编码也包含进去了，同理，全世界上百个国家，每个国家都有不同的编码标准，如果出现多种语言混合，就会出现乱码。

于是Unicode编码就出现，它把所有语言都统一到同一套编码系统里。

**比较ASCII和unicode编码**：

字母A用ASCII编码的十进制是65，二进制是0100 0001

汉字“中”的unicode编码是 20013，二进制是 0100 1110 0010 1101

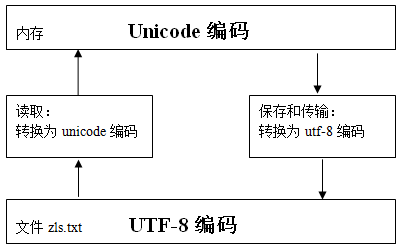
字母A的unicode编码只需要前面补0，即 0000 0000 0100 0001

用unicode虽然能解决乱码问题，但是如果内容全部是英文，unicode编码要比ASCII编码多出一倍的存储空间，同时传输的话也需要多一倍的带宽。

因此，出现了**可变长编码**“utf-8”，它把英文变长为一个字节，汉字变长3个字节，生僻字变长4~6个字节，如果传输大量的英文，则utf-8的作用就非常明显了。

由于unicode是统一长度的编码系统，在内存处理数据更方便，如果在内存中处理数据用utf-8 这种可变长编码系统，则非常复杂。

因此数据需要内存处理的时候就使用unicode编码，数据需要存储和传输的时候就使用utf-8编码。如下图所示：



打印系统编码

|  |
| --- |
| import sys  print(sys.getdefaultencoding()) |

注意：

在Python3中，所有字符在内存中都是统一用unicode编码的。

Python2如果字符串包含中文，那么在Windows系统中以gb2312编码，在Linux系统中则是以ascii方式编码存在。如果Python2的字符串不带 u’’ 则需要在encode之前，先decode，Windows用decode(“gb2312”)，Linux系统则用decode(“utf8”)，注意encode只能对unicode编码的字符串进行操作，在encode之前，必须把字符串都转换成unicode编码。