**pyspider学习**

# pyspider介绍

这个框架是一个国人编写的强大的网络爬虫系统并带有强大的WebUI。

采用Python语言编写，分布式架构，支持多种数据库后端，

强大的WebUI支持脚本编辑器，任务监视器，项目管理器以及结果查看器。

官方文档：http://docs.pyspider.org/en/latest/

中文文档：http://www.pyspider.cn

网络请求使用的是requests包，因此python环境中需要安装有requests

默认使用 sqlite 来保存文件和代码，创建的sqlite的路径是你运行 pyspider的目录下，创建一个data文件夹

譬如：C:\Users\51508>pyspider ， sqlite的数据库文件路径为 C:\Users\51508\data

# 测试运行

首先得安装 pyspider ，通过pip就可以安装

1、在命令行输入 pyspider all

2、在浏览器中访问 http://localhost:5000

3、整个页面分为两栏，左边是爬取页面预览区域，右边是代码编写区域。下面对区块进行说明：

左侧绿色区域：

这个请求对应的 JSON 变量，在 PySpider 中，其实每个请求都有与之对应的 JSON 变量，包括回调函数，方法名，请求链接，请求数据等等。

绿色区域右上角Run：点击右上角的 run 按钮，就会执行这个请求，可以在左边的白色区域出现请求的结果。

左侧 enable css selector helper: 抓取页面之后，点击此按钮，可以方便地获取页面中某个元素的 CSS 选择器。

左侧 web: 即抓取的页面的实时预览图。

左侧 html: 抓取页面的 HTML 代码。

左侧 follows: 如果当前抓取方法中又新建了爬取请求，那么接下来的请求就会出现在 follows 里。

左侧 messages: 爬取过程中输出的一些信息。

右侧代码区域: 你可以在右侧区域书写代码，并点击右上角的 Save 按钮保存。

右侧 WebDAV Mode: 打开调试模式，左侧最大化，便于观察调试。

def on\_start(self) 方法是入口代码。当在web控制台点击run按钮时会执行此方法。

self.crawl(url, callback=self.index\_page)这个方法是调用API生成一个新的爬取任务，这个任务被添加到待抓取队列。

def index\_page(self, response) 这个方法获取一个Response对象。 response.doc是pyquery对象的一个扩展方法。pyquery是一个类似于jQuery的对象选择器。

def detail\_page(self, response)返回一个结果集对象。这个结果默认会被添加到resultdb数据库（如果启动时没有指定数据库默认调用sqlite数据库）。你也可以重写on\_result(self,result)方法来指定保存位置。

@every(minutes=24\*60, seconds=0) 这个设置是告诉scheduler（调度器）on\_start方法每天执行一次。

@config(age=10 \* 24 \* 60 \* 60) age的单位是秒，这个设置告诉scheduler（调度器）这个request（请求）过期时间是10天，

10天内再遇到这个请求直接忽略。这个参数也可以在self.crawl(url, age=10\*24\*60\*60) 和 crawl\_config中设置。

@config(priority=2) 这个是优先级设置。数字越大越先执行。

# 实际代码测试

|  |
| --- |
| #!/usr/bin/env python  # -\*- encoding: utf-8 -\*-  # Created on 2018-03-14 21:44:04  # Project: v2ex\_project  from pyspider.libs.base\_handler import \*  class Handler(BaseHandler):  crawl\_config = {  }  @every(minutes=24 \* 60)  def on\_start(self):  self.crawl('https://www.v2ex.com/', callback=self.index\_page, validate\_cert=False)  @config(age=10 \* 24 \* 60 \* 60)  def index\_page(self, response):  # 取得本页面中 所有 a标签，属性 href 以 https://www.v2ex.com/?tab= 开头的 标签对象  # response.doc 是应用的 pyquery的语法  for link\_a in response.doc('a[href^="https://www.v2ex.com/?tab="]').items():  self.crawl(link\_a.attr.href, callback=self.tab\_page, validate\_cert=False)  @config(age=10 \* 24 \* 60 \* 60)  def tab\_page(self, response):  for link\_a in response.doc('a[href^="https://www.v2ex.com/go/"]').items():  self.crawl(link\_a.attr.href, callback=self.go\_page, validate\_cert=False)  @config(age=10 \* 24 \* 60 \* 60)  def go\_page(self, response):  # 爬取当前页面的所有帖子  for link\_a in response.doc('a[href^="https://www.v2ex.com/t/"]').items():  self.crawl(link\_a.attr.href, callback=self.detail\_page, validate\_cert=False)  # 翻页，获取所有页的内容  for link\_a in response.doc('a.page\_normal').items():  # 递归 爬取所有的页面，必须设置 age， 不然会无限死循环  self.crawl(link\_a.attr.href, callback=self.go\_page, validate\_cert=False)  '''  伪代码，也可以这样实现抓取所有页面  1、获取到最大的 页码 max\_pag  2、 for i in range(2, max\_pag+1):  url = 'https://www.v2ex.com/go/qna?p=' + str(i)  self.crawl(url, callback=self.go\_page, validate\_cert=False)  '''  @config(priority=2)  def detail\_page(self, response):  return {  "url": response.url,  "title": response.doc('title').text(),  }  # 用于接收这个 上面 的 return  def on\_result(self, result, response):  sql = 'insert into v2ex(url, title) values(%s, %s )'  insert(sql, (result['url'], result['title']))  import pymysql  '''  这个MySQL封装类，是没有实现连接池的，大家在项目的实际应用中。  需要使用连接池，使用pymysqlpool或者自己实现一个连接池  '''  """  db\_config是一些数据库的配置文件  """  class Mysql(object):  def \_\_init\_\_(self):  #数据库构造函数，从连接池中取出连接，并生成操作游标  self.\_conn = Mysql.\_\_getConn()  self.\_cursor = self.\_conn.cursor()  @staticmethod  def \_\_getConn():  """  @summary : 静态方法，从连接池中取出连接  @return MySQLdb.connection  """  config = {  'host': '127.0.0.1',  'port': 3306,  'user': 'root',  'password': '123456',  'db': 'v2ex',  'charset': 'utf8'  }  config['cursorclass'] = pymysql.cursors.DictCursor  conn = pymysql.connect(\*\*config)  return conn  def getAll(self, sql, param=None):  """  @summary: 执行查询，并取出所有结果集  @param sql:查询ＳＱＬ，如果有查询条件，请只指定条件列表，并将条件值使用参数[param]传递进来  @param param: 可选参数，条件列表值（元组 / 列表）  @return: result  list(字典对象) / boolean  查询到的结果集  """  if param is None:  count = self.\_cursor.execute(sql)  else:  count = self.\_cursor.execute(sql,param)  if count>0:  result = self.\_cursor.fetchall()  else:  result = []  return result  def getOne(self,sql,param=None):  """  @summary: 执行查询，并取出第一条  @param sql:查询ＳＱＬ，如果有查询条件，请只指定条件列表，并将条件值使用参数[param]  传递进来  @param param: 可选参数，条件列表值（元组 / 列表）  @return: result  list / boolean  查询到的结果集  """  if param is None:  count = self.\_cursor.execute(sql)  else:  count = self.\_cursor.execute(sql, param)  if count>0:  result = self.\_cursor.fetchone()  else:  result = False  return result  def getMany(self,sql,num,param=None):  """  @summary: 执行查询，并取出num条结果  @param sql:查询ＳＱＬ，如果有查询条件，请只指定条件列表，并将条件值使用参数[param]传递进来  @param num:取得的结果条数  @param param: 可选参数，条件列表值（元组 / 列表）  @return: result list / boolean 查询到的结果集  """  if param is None:  count = self.\_cursor.execute(sql)  else:  count = self.\_cursor.execute(sql, param)  if count>0:  result = self.\_cursor.fetchmany(num)  else:  result = False  return result  def insertOne(self,sql,value):  """  @summary: 向数据表插入一条记录  @param sql:要插入的ＳＱＬ格式  @param value:要插入的记录数据tuple / list  @return: insertId 受影响的行数  """  self.\_cursor.execute(sql, value)  return self.\_\_getInsertId()  def insertMany(self,sql,values):  """  @summary: 向数据表插入多条记录  @paramsql:要插入的ＳＱＬ格式  @paramvalues:要插入的记录数据tuple(tuple) / list[list]  @return: count受影响的行数  """  count = self.\_cursor.executemany(sql, values)  return count  def \_\_getInsertId(self):  """  获取当前连接最后一次插入操作生成的id, 如果没有则为０  """  self.\_cursor.execute("SELECT @@IDENTITY AS id")  result = self.\_cursor.fetchall()  return result[0]['id']  def \_\_query(self,sql,param=None):  if param is None:  count = self.\_cursor.execute(sql)  else:  count = self.\_cursor.execute(sql,param)  return count  def update(self,sql,param=None):  """  @summary: 更新数据表记录  @param sql: ＳＱＬ格式及条件，使用( % s, % s)  @param param: 要更新的值 tuple / list  @return: count 受影响的行数  """  return self.\_\_query(sql,param)  def delete(self,sql,param=None):  """  @summary: 删除数据表记录  @param sql: ＳＱＬ格式及条件，使用( % s, % s)  @param param: 要删除的条件 值 tuple / list  @return: count受影响的行数  """  return self.\_\_query(sql,param)  def begin(self):  """  @summary: 开启事务  """  self.\_conn.autocommit(0)  def end(self,option='commit'):  """  @summary: 结束事务  """  if option=='commit':  self.\_conn.commit()  else:  self.\_conn.rollback()  def dispose(self, isEnd=1):  """  @summary: 释放连接池资源  """  if isEnd==1:  self.end('commit')  else:  self.end('rollback')  self.\_cursor.close()  self.\_conn.close()  def insert(sql, params=None):  mysql = Mysql()  try:  ret = mysql.insertOne(sql, params)  finally:  mysql.dispose()  return ret  def insert\_many(sql, params=None):  mysql = Mysql()  try:  ret = mysql.insertMany(sql, params)  finally:  mysql.dispose()  return ret  def delete(sql, params=None):  mysql = Mysql()  try:  ret = mysql.delete(sql, params)  finally:  mysql.dispose()  return ret  def update(sql, params=None):  mysql = Mysql()  try:  ret = mysql.update(sql, params)  finally:  mysql.dispose()  return ret  def get\_one(sql, params=None):  mysql = Mysql()  try:  ret = mysql.getOne(sql, params)  finally:  mysql.dispose()  return ret  def get\_many(sql, params=None, num=None):  mysql = Mysql()  try:  ret = mysql.getMany(sql, num, params)  finally:  mysql.dispose()  return ret  def get\_all(sql, params=None):  mysql = Mysql()  try:  ret = mysql.getAll(sql, params)  finally:  mysql.dispose()  return ret |

# 框架分析

py2和py3兼容

1、database ： 针对数据库操作

2、fetcher：与task交互，获取信息，应用了lua嵌入脚本，使用了协程

3、libs：各种库，有多进程、测试、自定义Response等

4、message\_queue：消息队列

5、processor：项目管理的核心处理

6、result：处理返回结果，就是最后的 on\_result

7、scheduler：task队列调度

8、webui：html界面操作，使用tornado

9、run.py： 启动文件

# 错误处理

1、Exception: HTTP 599: SSL certificate problem: unable to get local issuer certificate

在 crawl 方法中 添加 validate\_cert=False 解决，这个类似requests的 s.verify = False

2、 rate/burst

rate ： 周期，单位秒，1秒执行几个task, 如 设置为 0.1 ，那么是 1秒执行0.1 个task ，设置为10，则是1秒10个task

burst：同时多少个task 执行