# 学习笔记

1. **生成ssh-keygen 密钥**

**ssh-keygen -t rsa -b 4096 -C** [**life88888@outlook.com**](mailto:life88888@outlook.com)

1. **pandas 转换日期**

**df[“日期”] = pd.to\_datetime(df[‘日期’], format=”%Y-%m-%s %H:%M:%S”)**

**总结**

* **将字符串转换为日期时间**：使用 pd.to\_datetime 函数。
* **指定日期时间格式**：通过 format 参数指定具体格式。
* **处理不标准日期时间格式**：使用 errors='coerce' 参数。
* **提取日期时间的组成部分**：使用 .dt 访问器。
* **格式化日期时间**：使用 strftime 方法。

1. **pandas 转换列名**

**df.rename(columns={“日期”: “datetime”, “代码”: “code”, ‘名称’: “name”}, inplace=True)**

**df.columns = [‘datetime’， ‘code’, ‘name’]**

**pandas 筛选数据**

**search\_code = df[df[‘code’] == 688615]**

**总结:**

**布尔索引：使用布尔条件进行筛选。**

**多个条件：使用 &、| 和 ~ 运算符组合多个条件。**

**query 方法：使用字符串表达式进行筛选。**

**loc 和 iloc：基于标签或位置进行筛选。**

**isin 方法：筛选包含在某个列表中的值。**

**notna 和 isna 方法：筛选非空或为空的值。**

**between 方法：筛选在某个范围内的值。**

1. **重写时间resample**

**resample\_df = df.resample(“15T”).last()**

**总结：**

* + **Y 年末频率**
  + **Q 季度末频率**
  + **M 月末频率**
  + **W 周频率**
  + **D 日频率**
  + **H 小时频率**
  + **T 分钟频率**
  + **S 秒频率**
  + **L 毫秒频率**
  + **U 微妙频率**

1. **apscheduler 定时器框架，简单示例**

from apscheduler.schedulers.blocking import BlockingScheduler

import datetime

def my\_task():

print(f"任务执行了: {datetime.datetime.now()}")

# 创建一个调度器

scheduler = BlockingScheduler()

# 添加一个每分钟执行一次的任务

scheduler.add\_job(my\_task, 'interval', minutes=1)

# 开始调度

try:

scheduler.start()

except (KeyboardInterrupt, SystemExit):

pass

1. **读取ini 文件**

import configparser  
  
  
def read\_conf\_ini(ini\_file\_name):  
 config = configparser.ConfigParser()  
 config.read(ini\_file\_name)  
 for section in config.sections():  
 print(f"section: {section}")  
 for key, value in config.items(section):  
 print(f"{key} = {value}")

1. **Apsheduler trigger 触发器**

**在 APScheduler 中，trigger 参数可以是一个对象，但这个对象必须是 APScheduler 提供的触发器类之一的实例。APScheduler 支持多种类型的触发器，比如 DateTrigger、IntervalTrigger 和 CronTrigger。这些触发器定义了任务何时以及如何被调度。**

**当你调用 add\_job 方法时，你可以传入一个触发器对象来指定任务的执行时间规则。以下是一些常见的触发器类型：**

* **DateTrigger: 用于只运行一次的任务，在特定的时间点运行。**
* **IntervalTrigger: 用于定期重复运行的任务，基于固定的时间间隔。**
* **CronTrigger: 类似于 Unix cron 作业，提供更复杂的调度规则。**

**下面是如何使用这些触发器对象的示例：**

1. **DateTrigger - 运行一次，在特定的时间点：**

**python**

**深色版本**

**from apscheduler.schedulers.blocking import BlockingScheduler**

**from apscheduler.triggers.date import DateTrigger**

**from datetime import datetime, timedelta**

**# 创建调度器**

**scheduler = BlockingScheduler()**

**# 定义将在未来5秒后运行的任务**

**run\_date = datetime.now() + timedelta(seconds=5)**

**trigger = DateTrigger(run\_date=run\_date)**

**# 添加任务**

**scheduler.add\_job(some\_callable, trigger=trigger)**

**# 启动调度器**

**try:**

**scheduler.start()**

**except (KeyboardInterrupt, SystemExit):**

**scheduler.shutdown()**

1. **IntervalTrigger - 基于固定时间间隔重复运行：**

**python**

**深色版本**

**from apscheduler.schedulers.blocking import BlockingScheduler**

**from apscheduler.triggers.interval import IntervalTrigger**

**# 创建调度器**

**scheduler = BlockingScheduler()**

**# 每隔5秒钟运行一次的任务**

**trigger = IntervalTrigger(seconds=5)**

**# 添加任务**

**scheduler.add\_job(some\_callable, trigger=trigger)**

**# 启动调度器**

**try:**

**scheduler.start()**

**except (KeyboardInterrupt, SystemExit):**

**scheduler.shutdown()**

1. **CronTrigger - 使用类似 cron 的表达式来调度任务：**

**python**

**深色版本**

**from apscheduler.schedulers.blocking import BlockingScheduler**

**from apscheduler.triggers.cron import CronTrigger**

**# 创建调度器**

**scheduler = BlockingScheduler()**

**# 每天早上6点运行的任务**

**trigger = CronTrigger(hour=6, minute=0)**

**# 添加任务**

**scheduler.add\_job(some\_callable, trigger=trigger)**

**# 启动调度器**

**try:**

**scheduler.start()**

**except (KeyboardInterrupt, SystemExit):**

**scheduler.shutdown()**

1. **logging 输出到控制台，和日志中**

import logging

# 创建 logger

logger = logging.getLogger('my\_logger')

logger.setLevel(logging.DEBUG) # 设置日志级别

# 创建 handler 输出到控制台

console\_handler = logging.StreamHandler()

console\_handler.setLevel(logging.DEBUG) # 控制台输出的日志级别

# 创建 handler 输出到文件

file\_handler = logging.FileHandler('app.log', encoding=”utf-8”)

file\_handler.setLevel(logging.DEBUG) # 文件输出的日志级别

# 创建 formatter 并设置格式

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - [%(filename)s:%(lineno)d] - [%(module)s.%(funcName)s] - %(message)s')

# 为 handler 添加 formatter

console\_handler.setFormatter(formatter)

file\_handler.setFormatter(formatter)

# 为 logger 添加 handler

logger.addHandler(console\_handler)

logger.addHandler(file\_handler)

# 记录一条日志

logger.info('这是一条信息级别的日志')

logger.error('这是一条错误级别的日志')

1. 爬取东方财富日频数据
2. Git 修改远程地址为ssh

git remote set-url origin [git@gitee.com:ShenKongLan/language-code-repos.git](mailto:git@gitee.com:ShenKongLan/language-code-repos.git)

1. 查看git 上传代码是使用http协议还是ssh协议

git remote -v

1. 请求限速

# 限速装饰器  
class RateLimiter:  
 def \_\_init\_\_(self, rate\_limit, period):  
 self.rate\_limit = rate\_limit # 每个周期最大请求数  
 self.period = period # 限速周期  
 self.tokens = self.rate\_limit  
 self.last\_refill\_time = datetime.now()  
  
 async def \_\_aenter\_\_(self):  
 while self.tokens <= 0:  
 await asyncio.sleep((self.period - (  
 datetime.now() - self.last\_refill\_time)).total\_seconds())  
 self.tokens = self.rate\_limit  
 self.last\_refill\_time = datetime.now()  
 self.tokens -= 1  
  
 async def \_\_aexit\_\_(self, exc\_type, exc, tb):  
 pass

1. asyncio 信号量使用

async def get\_five\_minutes\_hits():  
 if not os.path.exists("code.txt"):  
 get\_all\_stock\_code()  
 with open("code.txt") as f:  
 stock\_codes = json.loads(f.read())  
 # 设置并发数量  
 concurrency\_limit = 100  
 sem = asyncio.Semaphore(concurrency\_limit)  
 # 限速设置  
 # 每秒最多10个请求  
 rate\_limiter = RateLimiter(rate\_limit=10, period=timedelta(seconds=1))  
 # 创建一个任务队列  
 task\_queue = deque(stock\_codes)  
 dataframes = []  
 async with aiohttp.ClientSession() as session:  
 while task\_queue:  
 # 从队列中取出一个任务  
 stock\_code = task\_queue.popleft()  
 # 获取信号量  
 async with sem:  
 # 限速  
 async with rate\_limiter:  
 try:  
 result = await fetch\_stock\_detail(session, stock\_code)  
 df = await format\_data(result)  
 if isinstance(df, bool) and df is False:  
 continue  
 dataframes.append(df)  
 except Exception as e:  
 logger.error(traceback.format\_exc())  
 logger.error("Error fetching data for %s: %s",  
 stock\_code, e)  
 # 所有股票请求完成后, 保存到csv文件中  
 filename = "{}.csv".format(datetime.strptime(dataframes[0].iloc[0][  
 'datetime'],  
 "%Y-%m-%d "  
 "%H:%M").strftime(  
 "%Y%m%d"))  
 await save\_stock\_data(filename, pd.concat(dataframes))