代码脚手架使用说明文档

## 为什么我们需要一套脚手架？

编程领域中的“脚手架（Scaffolding）”指的是能够快速搭建项目“骨架”的一类工具，可以理解为一套项目模板，包含一些在编写项目时重复性使用的通用工具。

能帮助我们解决一些痛点问题：

* + 项目结构不统一、不规范
  + 项目配置越来越繁琐、耗时，重复
  + 代码可读性增强

## 项目结构



项目主目录下包含四个文件夹：doc（存放相关文档），code(代码主体部分)，data(数据存储)，log（日志）

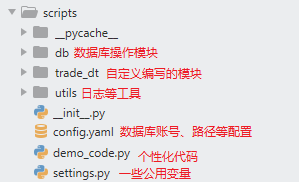
## Code部分功能

### main.py

建议只在外部暴露main一个主程序调用接口



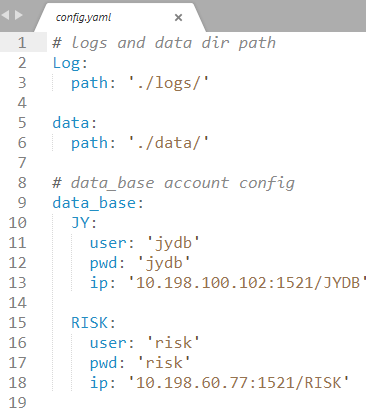
### -scripts



#### Config.yaml

配置一些路径如log路径、data路径（主程序下的相对路径）

数据库账号配置（会在db模块被调用）



#### Settings.py

一些公用变量（读取config内容等），可能被反复调用的变量等

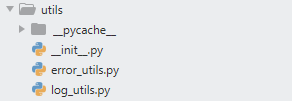


#### Demo\_code.py

项目内自己个性化编写的代码部分



#### -utils

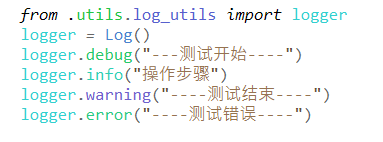


##### log\_utils

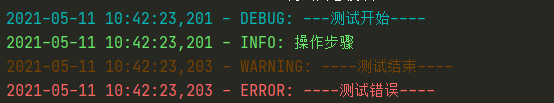
日志工具，使用方法见demo\_code.py，在代码运行过程中记录日志并自动存储在logs目录下，同时在工作台打印不同级别log信息



调用方式：以scripts目录下py文件为例：



工作台输出：



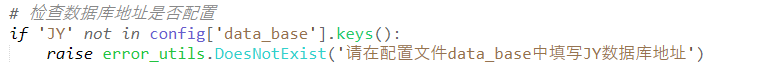
同时，会在logs文件夹下生成当天日志：



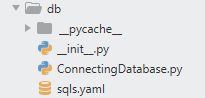
##### Error\_utils.py

Error工具，可自定义error类型

使用示例：import后



##### db



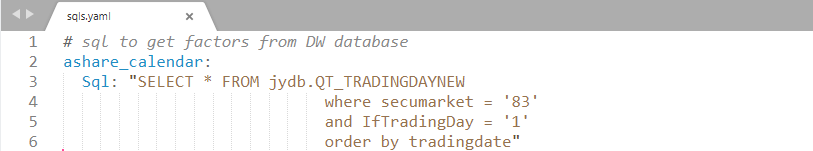
###### ConnectingDatabase.py

在之前王沛基础上做了一点小小的调整，即在init时增加一项账户登录信息（可从config.yaml中统一设置和读取）



###### Sqls.yaml

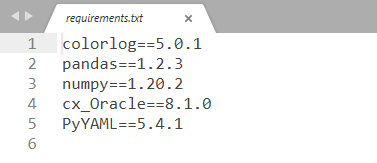
统一存放所有sql语句，并在ConnectingDatabase加载（yaml中不支持中文字段）



使用方式（见Demo\_code.py）：



# Requirement.txt



记录所有依赖包及其精确的版本号，以便新环境部署。

生成方式

# 安装

pip install pipreqs

# 在当前目录生成

pipreqs . --encoding=utf8 --force

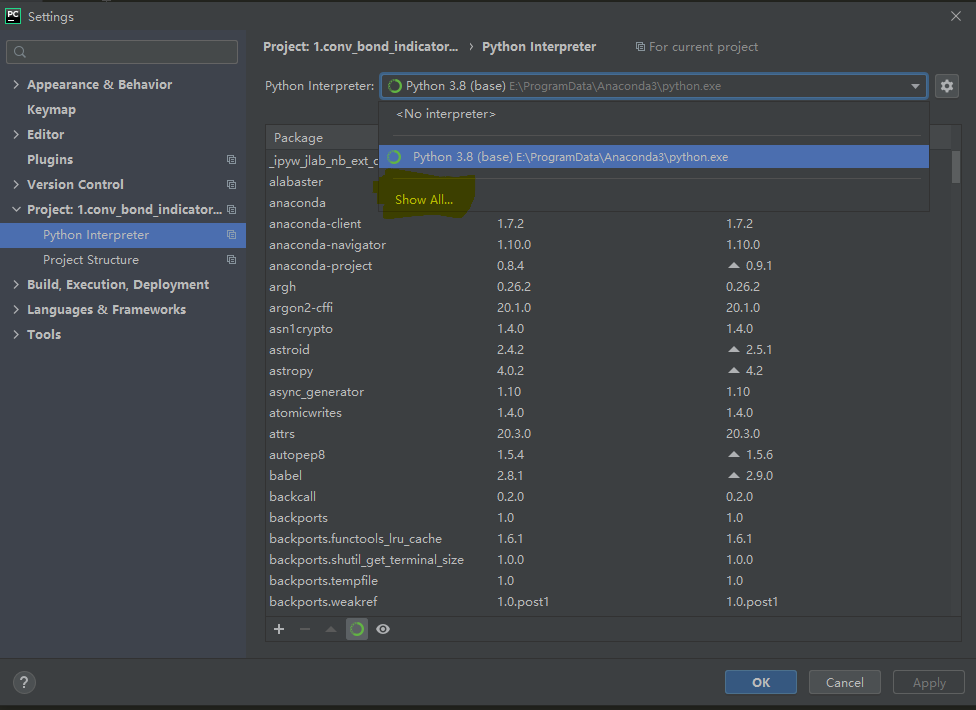
# 附录

## 虚拟环境搭建

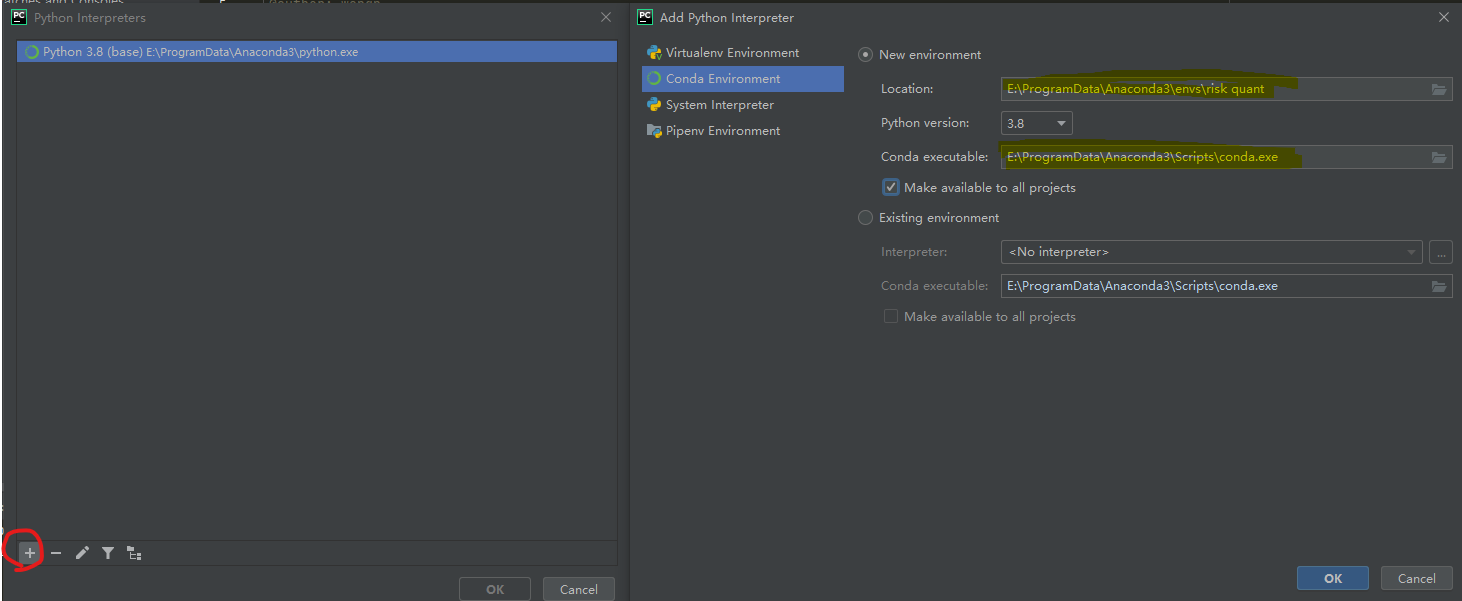
当代码在不同电脑上运行时，由于每个人python包的版本不同，运行时可能导致版本错误，可以考虑构建新的虚拟环境，在环境中根据项目依赖的版本号（见上文requirement.txt）进行版本配置。隔离的虚拟环境不会影响本地原始环境。

#### 新建虚拟环境

Setting-Python Interpreter-Show All

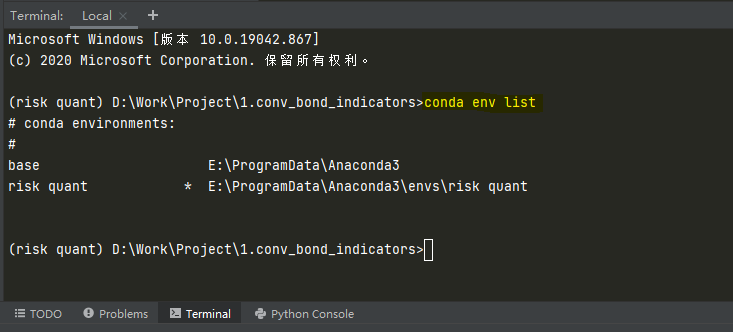


点击+号，选择环境地址（一般放在Anaconda目录envs下，risk quant为自己起的环境名称）

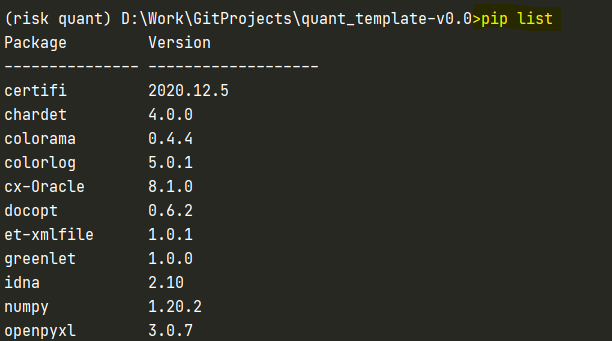


#### 查看已有的环境

在命令行运行 **conda env list**可查看系统中已有的虚拟环境



#### 查看已有的环境中python包的版本



#### 注意

在全新的环境运行代码时，可能需要同步调整anaconda目录下wind python插件和oracle数据库配置（ddl文件等）