

|  |
| --- |
| **中信证券**  **智能投资策略部署手册** |

目录

**[中信证券](#_Toc61522634)** [1](#_Toc61522634)

**[智能投资策略部署手册](#_Toc61522635)** [1](#_Toc61522635)

[一、 前言 3](#_Toc61522637)

[二、 策略逻辑 3](#_Toc61522638)

[三、 硬件清单 6](#_Toc61522639)

[四、 安装条件及策略部署 6](#_Toc61522640)

[（一） 安装的前提条件 6](#_Toc61522641)

[（二） 代码包 6](#_Toc61522642)

[（三） 策略部署 6](#_Toc61522643)

[五、 策略回测 12](#_Toc61522644)

[（一）概述 12](#_Toc61522645)

[（二）回测使用案例 12](#_Toc61522646)

# 前言

本文档为智能投资策略部署手册，主要内容包括对策略逻辑、硬件、软件清单、安装部署等进行介绍，供系统运维人员、开发人员、测试人员使用。

# 策略逻辑

**（一）选股逻辑**

选股是根据配置的选股条件，筛选出每日标的池，其主要逻辑如下：

1、回测部分

（1）配置选股条件

根据选股策略id，数据库配置对应选股条件。

（2）选股条件

每个选股条件将对于全市场进行所配参数筛选，得到结果存入redis（{date:list(code}格式）、mysql中。

（3）选股结果

根据选股id得到每个选股条件。选股时先从redis获取条件数据，若无则从mysql获取，若mysql无则调用选股条件函数从wind原库获取，每个条件取交集得到最终每日选股结果存入redis（{date:list(code)}格式）、mysql中。

2、实盘部分

与回测逻辑相同，区别在于实盘会增加禁买池的筛选。选股结果会去掉禁买池的标的，并将当日禁买的标的落入当日黑名单。

**（二）资产配置**

根据策略id得到资产配置方式，根据资产配置方式的不同，读取市盈率、流通市值、波动率、昨收盘等不同的指标数据，通过模型计算，为选股结果分配持仓，保证当天的持仓比例总和为1.

**（三）风控逻辑**

风控是提供通过止损，最大回撤，集中度等来控制仓位或个股占比的功能，其主要逻辑如下：

1、回测部分

（1）止损

根据策略当前净值来判断是否达到风控设定的止损阀值，达到就将仓位降到风控配置的仓位值

（2）最大回撤

本策略n天内最大回撤或者n天内配置的基准标的（一般为指数）的最大回撤来控制仓位

（3）个股集中度

通过配置个股最大占比来控制仓位

当上述风控配置有多个时，最终仓位将会取每个风控类型的目标仓位中的最小值

2、实盘部分

昨日总资产获取来源不同，其他逻辑与回测相同

**（四）生成订单**

生成订单部分是根据目标仓位，当前总资产和当前持仓信息，计算出今日的调仓指令，和成交详情，目前支持类型（股票，ETF），主要逻辑如下：

1、回测部分

（1）根据策略配置中的撮合价格类型获取预成交价（price），并按此预成交价重新计算totalAsset

（2）结合各成分股目标仓位占比（targetratio）和总资产计算出目标占资（targetNotional）

（3）用各股的目标占资（targetNotiional）- 持仓市值 ，计算出需调仓金额（resultNotional），负值，需要卖出，正值，需要买入

（4）根据标的及策略配置的交易系统获取计算出相应的手续费，然后从totalasset中减去手续费,并重新计算买的需调仓金额（resultNotional）

（5）用 resultNotional/price 计算出基础调仓量。

（6）对基础调仓量根据买卖不同方式做整百处理，基本处理方式：

对于买，以该标的类型的最小下单量(min\_order\_volume)为基准做向下取整，

对于卖，以该标的类型的最小下单量(min\_order\_volume)为基准向上取整

2、实盘部分

与回测逻辑相同，不同之处在于 总资产（totalAsset） 和持仓（preposition）是通过交易接口实时获取的最新状态。price从交易系统获取最新价

**（五）交易执行**

交易执行主要指订单的执行情况。策略配置执行目标价参数，可以选择以开盘价、收盘价、均价等价格执行订单。回测及模拟部分以设置的目标价全部成交；实盘部分成交价和成交量通过算法交易的实际执行情况为准，根据不同的目标价参数选择相应的算法。

**（六）策略清算**

策略清算指每一个交易日对策略进行交易、持仓、账户、资产、绩效等方面的清算过程。

其主要流程如下：

1、交易情况

交易情况包括交易标的所属策略、交易日期、交易标的、买卖方向、多空方向、实际占资、成交量、成交价、交易费用、名义本金及交易盈亏。成交情况参考上述（五）交易执行部分。其中，

实际占资 = 名义本金 \* 保证金率

交易费用 = 成交量 \* 成交价 \* 费率

交易盈亏 = 多/空 \* 买/卖 \* 成交量 \* （今日收盘价（期货是结算价） - 成交价）

2、持仓情况

持仓情况包括交易标的所属策略、交易日期、多空方向、交易标的、账户类型、实际占资、持仓量、名义本金、标的前一交易日收盘价、标的今日收盘价及持仓盈亏。其中，

持仓盈亏 = 多/空 \* 买/卖 \* 持仓数量 \* （今日收盘价（期货是结算价） - 昨日收盘价（期货是结算价））

3、账户情况

账户情况为分账户统计策略情况，账户类型包括期货账户和现货账户，具体包括交易标的所属策略、交易日期、账户类型、持仓市值、现金、日终总资产、日初总资产。有期货交易标的时，现金全部放在期货账户，具体如下：

（1）无期货持仓时，账户类型仅为现货类型：

现金 = 策略今日盈亏 + 前一交易日日终总资产 – 今日持仓市值

日终总资产 = 策略今日盈亏 + 前一交易日日终总资产

（2）有期货持仓时，账户类型为期货和现货类型：

期货：

现金 = 策略今日盈亏 + 前一交易日日终总资产 – 今日持仓市值

日终总资产 = 现金 + 前一交易日期货日终总资产

现货：

现金 = 0

日终总资产 = 现货持仓市值

4、资产情况

资产情况为策略合计情况，具体包括交易标的所属策略、交易日期、持仓市值、现金、日终总资产、日初总资产、策略总盈亏、策略净值。其中，

策略总盈亏 = 策略今日盈亏 + 前一交易日策略总盈亏

策略净值 = (日终总资产/日初总资产) \* 前一交易日策略净值

5、绩效情况

绩效指标从多个角度对策略进行评价，绩效指标包括累计收益率、年化利率-复利、年平均收益率-非复利、夏普率、索提诺率、最大回撤率、最大回撤时长、5交易日最大回撤、20交易日最大回撤、日平均收益率-非复利、日平均波动率。绩效评价周期涵盖近一个月、近三个月、近半年、近一年、今年以来、成立以来。

# 硬件清单

|  |  |  |
| --- | --- | --- |
| 硬件IP | 用途 | 硬件配置 |
| 172.23.122.17 | 智能投资定时任务 | 内存：128G |

# 安装条件及策略部署

## （一） 安装的前提条件

操作系统： Ubuntu 20.04

## （二） 代码包

|  |  |
| --- | --- |
| 代码包 | 说明 |
| ai-investment-manager | 智能投资策略代码 |

## （三） 策略部署

**1、安装gitlab-runner**

如已配置gitlab-runner需要新注册一个runner，只需执行第（7）步

（1）下载

sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

（2）修改权限

sudo chmod +x /usr/local/bin/gitlab-runner

（3）创建一个gitlab-runner用户：

sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

（4）安装和配置home文件夹，此命令会在/home目录创建gitlab-runner用户的文件：

sudo gitlab-runner install --user=gitlab-runner –working-directory=/home/gitlab-runner

（5）修改gitlab-runner用户的密码：

sudo su root

passwd gitlab-runner

输入新gitlab-runner用户的新密码

（6）进行gitlab-runner用户的ssh免密码登录设置：

su gitlab-runner

git init

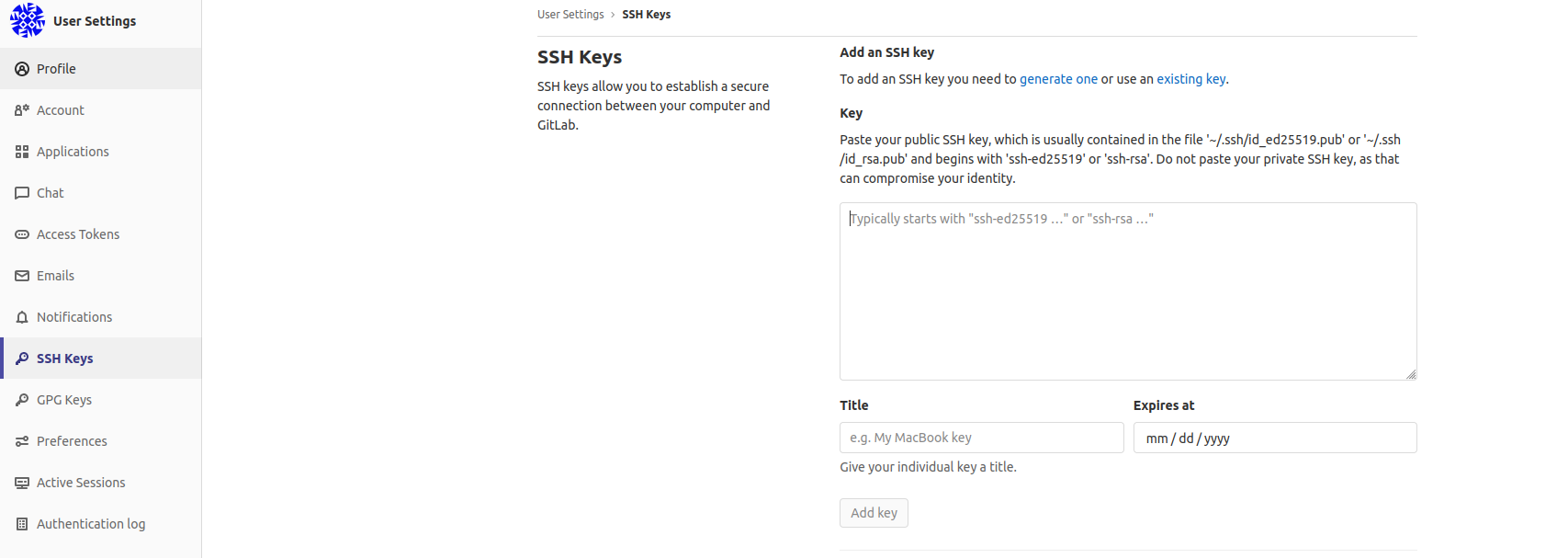
git config --global user.name '用户名'

git config --global user.email '用户邮箱'

其中用户名和邮箱任意

ssh-keygen -t rsa

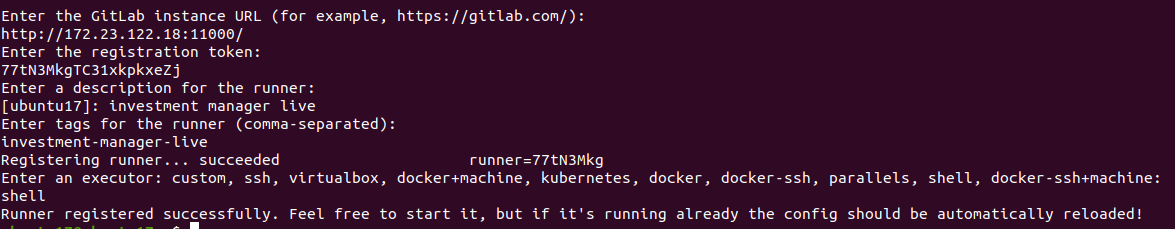
cat ~/.ssh/id\_rsa.pub 内容复制到gitlab设置ssh里，见下图右侧

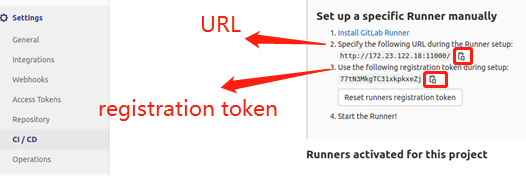


（7）注册gitlab-runner：

切换为非gitlab-runner用户

sudo gitlab-runner register，根据提示输入对应项目的配置，见下图。URL和registration token根据下图进行复制，tags自己设定，但需与yml文件（下述）tags保持一致，executor选择shell即可





（8）设置并发数

vim /etc/gitlab-runner/config.toml

第一行为concurrent数量，默认是1，修改成需要的数字保存

（9）启动runner

sudo gitlab-runner start（如果start失败，可执行sudo gitlab-runner verify然后sudo gitlab-runner restart）。

**2、编辑yml脚本**

在项目根目录下创建一个gitlab-ci.yml文件（文件名不能更改），以下为参考范例，实现的功能是拉取目标分支代码，切换到某一个tag（以version参数值为准），运行dailyTask.sh，参数为PARAM，version和PARAM参数设定详解见第4步（自动化部署）：

stages:  
 - test  
job1:  
 stage: test  
 tags:  
 - investment-manager  
 only:  
 - /^r-.\*$/   
script:  
 - cd /home/gitlab-runner/ai-investment-manager/  
 - echo $CI\_COMMIT\_REF\_NAME  
 - git checkout $CI\_COMMIT\_REF\_NAME  
 - git pull  
 - git checkout $version  
 - bash dailyTask.sh $PARAM

参数详解：

stages:定义pipeline的全部阶段（stage），阶段内所有任务并行执行，全部执行成功开始下一阶段任务，任何阶段内任意job执行失败都会导致pipeline失败，所有stage，job执行成功后pipeline会显示pass。如果未定义stages，则默认有build、test、deploy三个阶段，如果未定义stage，则默认test阶段

job1:任务名称，可自由定义

stage:关键字，用于指定任务在什么stage运行；

tags:关键字，用于关联上一步设置的gitlab-runner，应对多个gitlab-runner任务匹配问题

script：关键字，按顺序撰写该任务的shell脚本；

only：关键字，用于指定依赖的代码分支，可支持正则表达式；

**3、安装根网依赖**

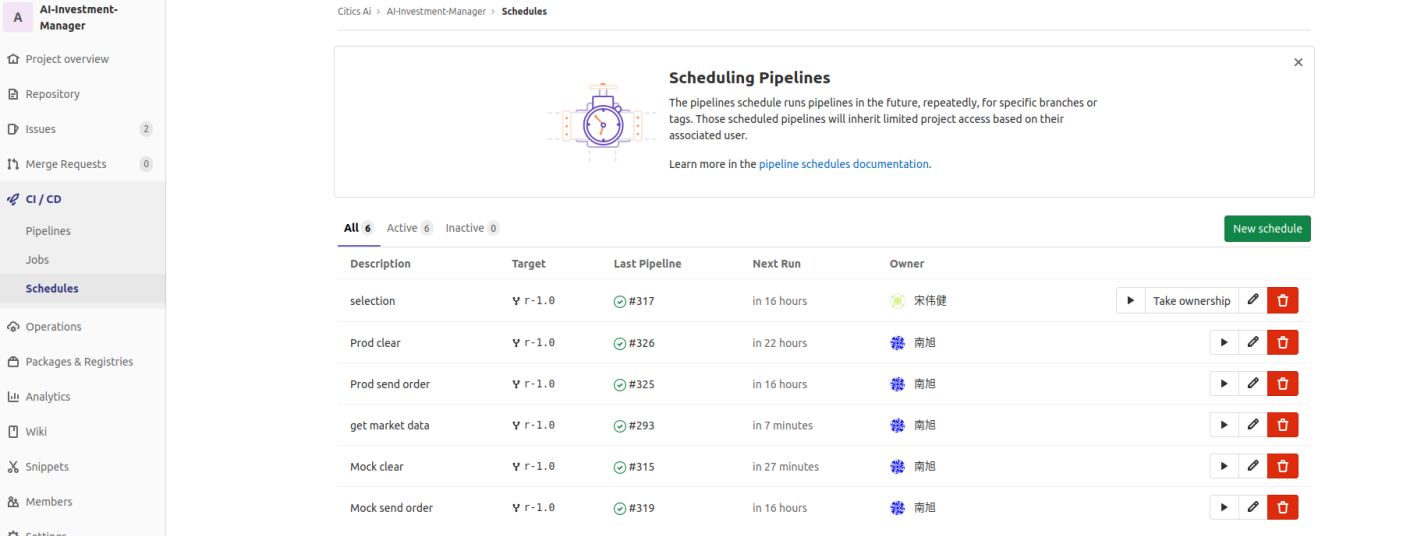
根网依赖位置在ai-investment-manager包里，路径为

ai-investment-manager/tradingSystem/rootNet/rootNetInstall

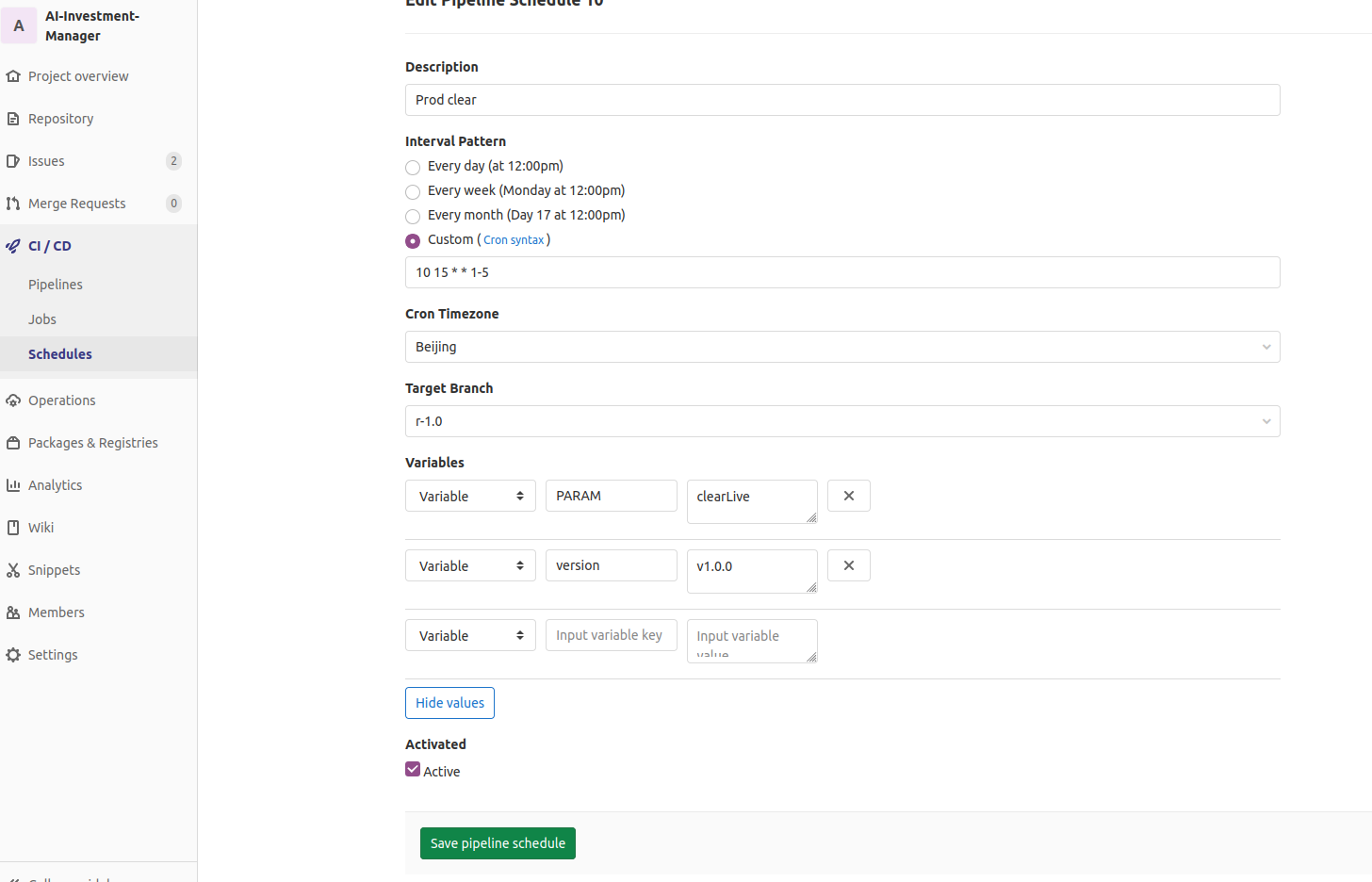
进入上述路径并执行，python3 setup.py install

**4、自动化部署**

（1）进入CI/CD目录下Schedule界面，点击New schedule

（2） 设置参数

范例见下图

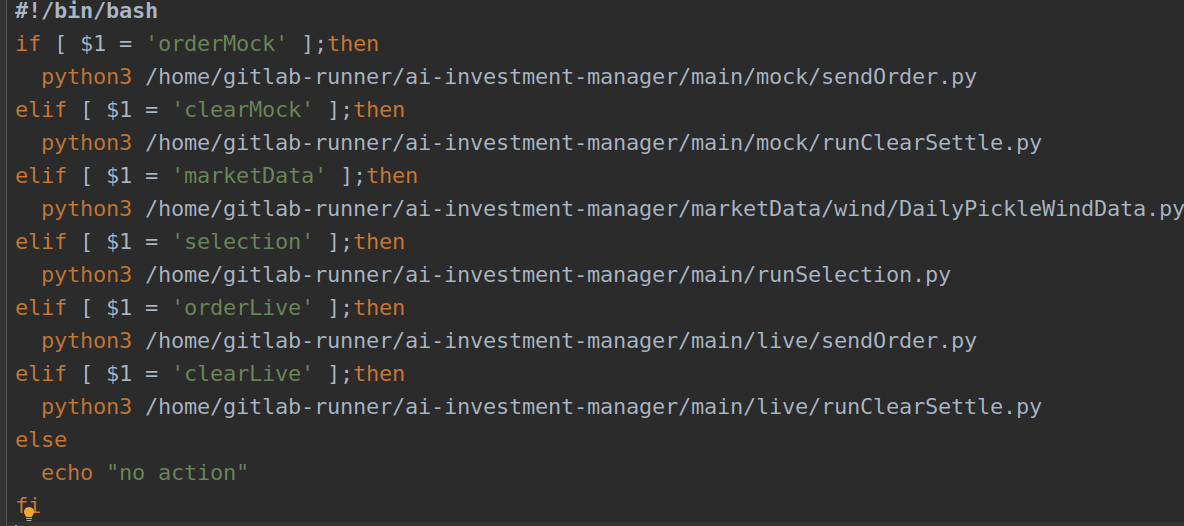


interval Pattern 为定时任务运行时间

Cron Timezone为时区，选Beijing

Target Branch为目标分支，作用是运行目标分支的yml文件

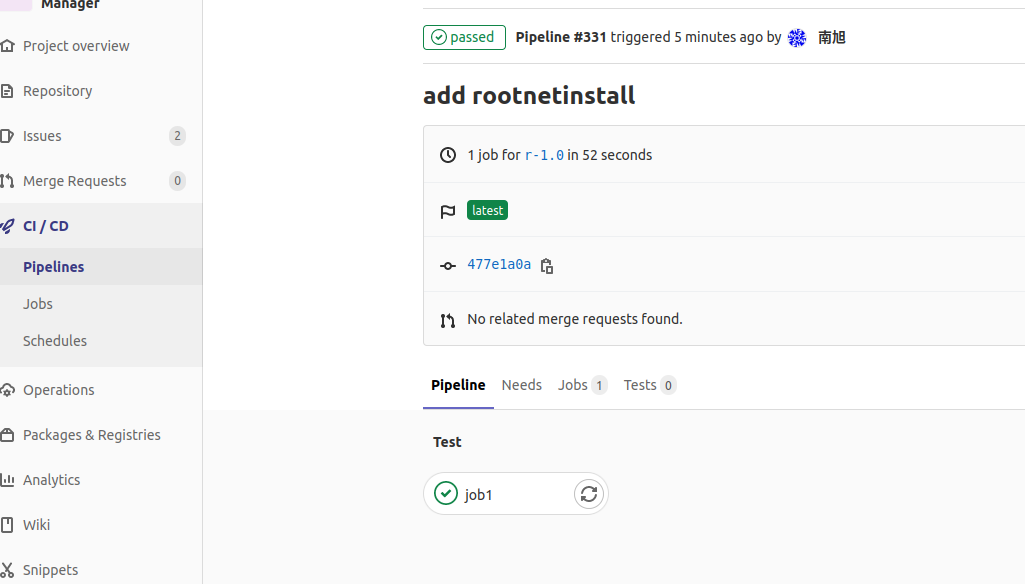
Variables 指为yml传入的参数，第一列为变量类型，第二列为变量名，第三列为变量值，version为相应tag版本，PARAM参数代表运行不同的脚本，对应关系见dailyTask.sh，可自行编写，范例见下图



Activated 表示是否运行定时任务

部署完成点save pipeline schedule

（3）任务查看点击（1）步中的last pipeline列，可以看到对应任务的执行情况，见下图，点击job1可以通过终端显示查看代码运行情况



**5、定时任务汇总**

（1）每日9：00 运行选股脚本

（2）每日9：26 运行生产环境订单脚本

（3）每日9：27 运行模拟环境订单脚本

（4）每日15：10 运行生产环境清算脚本

（5）每日17：00 运行落当日行情数据脚本

（6）每日17：20 运行模拟环境清算脚本

# 策略回测

## （一）概述

策略回测是包含选股、资产配置、风控、生产订单、交易执行、清算和绩效评估全流程的模块，回测模块通过利用redis和内存进行数据存储提高数据使用效率，通过矩阵思维同时处理多个策略优化计算效率，通过进程、线程提高资源利用效率，最终优化回测速度。

## （二）回测使用案例

1、挂载本地数据

（1）第一次挂载：

进入ai-investment-manager包所在目录，运行以下命令

sh ai-investment-manager/nfs\_start.sh

（2）非第一次挂载或199服务器重启：

进入ai-investment-manager包所在目录，运行以下命令

sudo umount /share\_data

sudo mount –t nfs 10.24.206.199:/share\_data /share\_data

2、运行回测

进入ai-investment-manager包所在目录，运行以下命令

python3 ai-investment-manager/main/backtest/runBacktest.py

3、回测类Backtest使用说明如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 初始化回测类 | | | |
| def \_\_init\_\_(self, strategy\_ids, start\_date, end\_date, env=’dev’) | | | |
| 名称 | 类型 |  | 说明 |
| strategy\_ids | list | 输入 | 策略id的list，如[‘turing\_1-3’,’turing\_1-4’] |
| start\_date | str | 输入 | %Y%m%d格式，如’20210112’ |
| end \_date | str | 输入 | %Y%m%d格式，如’20210112’ |
| env | str | 输入 | 回测默认在dev环境 |
| 返回值 | 无 | 返回值 | 无 |
| 初始化asset数据 | | | |
| def \_setAsset (self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | dataframe | 返回值 | 策略配置首日asset数据 |
| 初始化account数据 | | | |
| def \_setAccount (self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | dataframe | 返回值 | 策略配置首日account数据 |
| 初始化股票池 | | | |
| def \_setSecurityPool (self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | 无 | 返回值 | 无 |
| 初始化目标持仓占比 | | | |
| def \_setTargetPosition (self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | 无 | 返回值 | 无 |
| 初始化策略配置参数 | | | |
| def \_getStrategyConfig(self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | List(dict) | 返回值 | 返回策略配置字典列表，如  [{‘strategy\_id’:’turing’,’selection\_id’:’1’},  {‘strategy\_id’:’turing-1’,’selection\_id’:’2’}] |
| 初始化策略选股、配置、风控、绩效等参数 | | | |
| def \_getConfig(self, table, key) | | | |
| 名称 | 类型 |  | 说明 |
| table | str | 输入 | 策略配置表名，如’risk\_config’,’allocation\_config’ |
| key | str | 输入 | 相应配置id,如风险配置为’risk\_id’，选股配置为’selection\_id’ |
| 返回值 | dict（dict） | 返回值 | 返回字典套字典，如  {’turing’:{selection\_id’:’1’},{’turing-1:{’selection\_id’:’2’}} |
| 初始化wind数据 | | | |
| def \_getConfig(self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | 无 | 返回值 | 无 |
| 回测结果存入数据库 | | | |
| def saveResult2DB(self, tables ,condition) | | | |
| 名称 | 类型 |  | 说明 |
| tables | list | 输入 | 要存入数据库的表名，不写则全存 |
| conditions | str | 输入 | sql语句，是依据绩效指标判断是否存数据库的条件，如’total\_returns>1000’，指只有策略的总收益大于1000%才存数据 |
| 返回值 | 无 | 返回值 | 无 |
| 回测主程序 | | | |
| def run(self) | | | |
| 名称 | 类型 |  | 说明 |
| 返回值 | 无 | 返回值 | 无 |