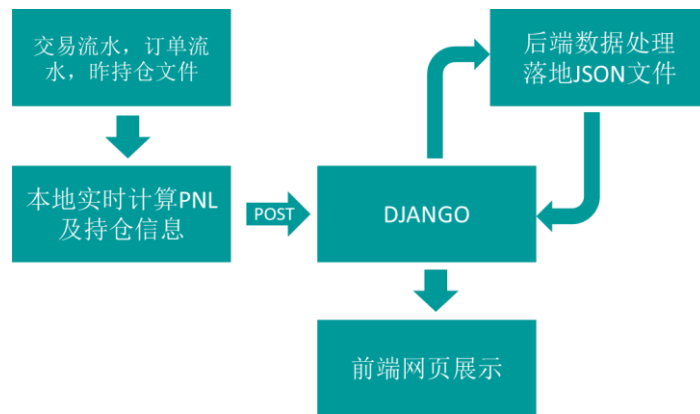


目录

Mysite 开发说明文档	1
1. Mysite 设计模式	1
2. Django 服务的启动与运行	1
3. Django 框架及重要文件	1
3.1 Django 框架介绍	1
3.2 重要文件	2
4. URL 分发器	3
5. Web 结构详解及后端代码	5
5.1 登录	5
5.2 实时数据模块	6
5.3 投研分析	6
5.4 策略管理	6
5.5 主页	7
5.6 历史净值	7
6. JS 脚本及前端代码	8
6.1 JS 脚本所在路径	8
6.2 JS 脚本介绍	8
7. HTML 详解及注意事项	11
7.1 HTML 模板所在路径	11
7.2 HTML 介绍	11
8. 定时任务管理	12
9. POST-Sender	12
10. 常见的问题及处理方法	13

Mysite 开发说明文档

1. Mysite 设计模式



2. Django 服务的启动与运行

找到../mysite/manage.py 所在文件路径，双击 run.bat 或者在该路径下启动 cmd，并执行

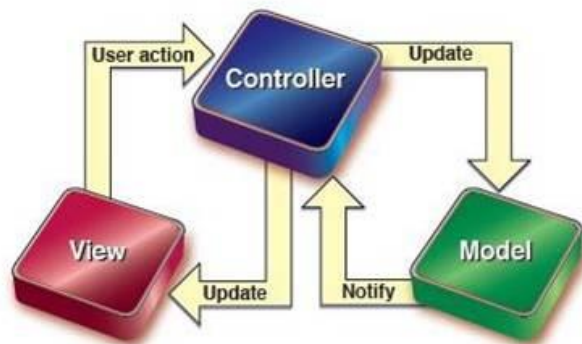
```
python manage.py runserver 0.0.0.0:50100
```

其中 50100 是想要启动 Django 所在的端口

3. Django 框架及重要文件

3.1 Django 框架介绍

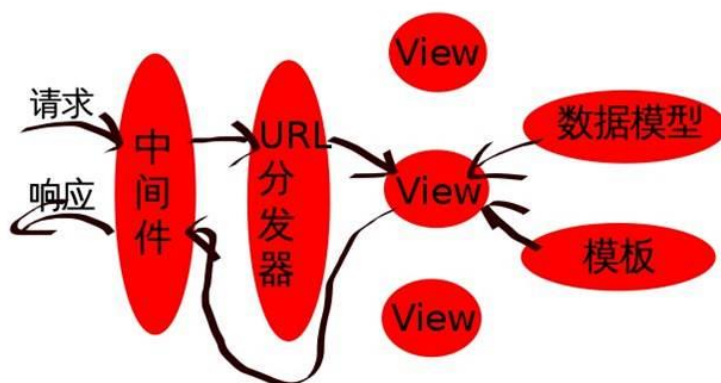
首先说说 Web 服务器开发领域里著名的 MVC 模式, 所谓 MVC 就是把 Web 应用分为模型 (M), 控制器 (C) 和视图 (V) 三层, 他们之间以一种插件式的、松耦合的方式连接在一起, 模型负责业务对象与数据库的映射 (ORM), 视图负责与用户的交互 (页面), 控制器接受用户的输入调用模型和视图完成用户的请求, 其示意图如下所示:



Django 的 MTV 模式本质上和 MVC 是一样的，也是为了各组件间保持松耦合关系，只是定义上有些许不同，Django 的 MTV 分别是值：

- M 代表模型 (Model)：**负责业务对象和数据库的关系映射(ORM)。
- T 代表模板 (Template)：**负责如何把页面展示给用户(html)。
- V 代表视图 (View)：**负责业务逻辑，并在适当时候调用 Model 和 Template。

除了以上三层之外，还需要一个 URL 分发器，它的作用是将一个个 URL 的页面请求分发到不同的 View 处理，View 再调用相应的 Model 和 Template，MTV 的响应模式如下所示：



- 1 Web 服务器（中间件）收到一个 http 请求
- 2 Django 在 URLconf 里查找对应的视图(View)函数来处理 http 请求
- 3 视图函数调用相应的数据模型来存取数据、调用相应的模板向用户展示页面
- 4 视图函数处理结束后返回一个 http 的响应给 Web 服务器
- 5 Web 服务器将响应发送给客户端

3.2 重要文件

Django 框架下有一些重要的 py 文件，此外还有我们自己写的一些重要文件也会在这个章节进行解释说明，此处仅选取几个重要的文件进行介绍：

- 路径配置文件：datapath.py
- 网页应用文件：apps.py
- URL 分发器：urls.py

视图文件: `views.py`
定时任务文件: `task.py`
设置文件: `settings.py`
模型文件: `models.py`

以下将逐一对其进行简单说明。

`datapath.py`

保存重要的**路径配置**信息, 包括 HTML 模板路径, 临时数据路径, 历史净值数据路径, 数据备份路径现货投研分析数据路径及 Django 策略管理路径。

`apps.py`

保存 Django 管理的所有应用, 起名为 `interface` (界面), 注意: 在部署 Django 到新服务器的时候, 需要对应用执行 `migrate` 操作, `django` 会导入每个 `app` 下的 `model`, 默认会搜索 `app` 包下的 `models.py`。

`urls.py`

URL 分发器: 管理应用下的所有 `url`, 其中包括 `POST` 接口, 及 `web` 网页链接及前端响应函数, 详情请看第 4 节: URL 分发器。

`view.py`

是最重要的 `py` 脚本, 包含了所有的后端数据操作及**前端响应函数**, `POST 接口函数`对所有请求进行处理的场所, 并在适当时候调用 `Model` 和 `Template`, 是与用户进行交互的地方。

`task.py`

为**定时任务**脚本, 使用装饰器装饰 `django-apscheduler` 的定时任务, `django-apscheduler` 本身也是一个 `app`, 所以要在 `settings.py` 中 `INSTALLED_APPS` 中加入这个 `app`。(详情请查看定时任务管理)

`setting.py`

这个是 Django 重要的**设置**文件, 自己开发的应用及需要用到的 Django 应用包都需要在 `INSTALLED_APPS` 添加进去, 并在安装 Django 时进行 `migration` 迁移 (`python manage.py migrate` 详见 Django 安装说明文档) `ALLOWED_HOSTS = ['*']` 意思是允许任何 IP 地址访问

`models.py`

Django 的 ORM 模式, 会把数据库的表 `Table` 映射成类对象, 数据库中的所有表均可以转换成类, 使用类似于类调用自身属性与方法的使用方法。

4. URL 分发器

如上文提到的 Django 的 MTV 模式, 在 Django 中, URL 分发器的作用是将一个个 URL 的页面

请求分发给不同的 View 处理, View 再调用相应的 Model 和 Template。本项目中的 URL 都保存在../mysite/mysite/urls.py 中, 主要有如下三类:

1. **Html 网页响应函数**。如: 主页 main/, 历史净值网页 netvalue/。
2. **网页交互式响应函数**。如: 添加策略 add_strategy/为策略管理页面提交新策略的响应函数。
3. **数据 API 函数**。包括 POST 与 GET 两种 API 函数。如: posttotalpnl/为接收其他服务器 post 过来的 pnl 数据并在后端中对数据进行修改的接口函数, newOrderApi/即为 GET 方法的 API 函数。

url 详情, 请看下图中的分类:

```
urlpatterns = [

    # HTML网页响应函数
    path('main/', main), # Web主页
    path('real-time/', real_time), # 实时数据
    path('quant_simulate/', quant_simulate), # 量化组模拟
    path('quant_real/', quant_real), # 量化组实盘
    path('zhang_simulate/', zhang_simulate), # 张总模拟盘
    path('zhang_real/', zhang_real), # 张总实盘
    path('lifeng/', lifeng), # 李峰模拟盘
    path('cheng/', cheng), # 程总实盘
    path('lifeng_netvalue/', lifeng_netvalue), # 李峰策略历史净值
    path('research/', research), # 现货投研
    path('netvalue/', netvalue), # 历史净值
    path('add_user/', add_user), # 添加用户
    path('login/', login), # 登录
    path('logout/', logout), # 退出登录
    path('concat/', concat_us), # 联系我们
    path('manage/', management), # 策略管理

    # 网页交互式响应函数
    path('deletejson/', deletejson), # 删除订单或持仓
    path('stopstrategy/', stopStrategy), # 策略控制 (未启动)
    path('result/', tanchuang), # 弹窗
    path('add_product/', add_product), # 添加产品
    path('add_account/', add_account), # 添加产品-账号
    path('add_strategy/', add_strategy), # 添加策略
    path('change_strategy/', change_strategy), # 修改策略配置
    path('add_user_success/', add_user_success), # 添加用户成功
```

```

# API接口函数
path('postPnlPositionOrder/', post_all_pos_pnl_order),
# 一次性POST所有信息, 包括PNL, 持仓详情及新订单详情
path('posttotalpnl/', posttotalpnl), # POST PNL
path('posttotalpnl_makeup/', posttotalpnl_makeup), # PNL重传
path('postorder/', postorder), # POST 新订单
path('postpositiondetail/', postpositiondetail), # POST 持仓详情
path('postanything/', postAnything), # POST 任意信息
path('postCurrProdTs/', postCurrentProductTs), # POST 现货最新价格
path('postCurrentProductReturn/', postCurrentProductReturn), # POST 现货涨跌幅
path('netvalue_hand_clear/', clearHistoryPNL), # 手动净值清算
path('newOrderApi/', get_new_order), # 从Django中获取当前最新订单
path('refine_order/', refine_order),
# 筛选提炼订单 (仅保留订单状态是“已成”, “已撤单”或“拒绝”的订单, 返回给客户端)

# 其他
path('testApi/', testApi), #
path('admin/', admin.site.urls), #
path('ctimes/', ctimes), #
path('interface/', interface), #
# path('GetStrategyList/', GetStrategyList),
path('testtop/', testtop), #
path('socket/', socket), #
path('add/', add) #

] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

5. Web 结构详解及后端代码

5.1 登录

对应 `interface.templates.login.html` 及 `view.login` 函数, 在登录时会在浏览器 `session` 中记录登录名及登录状态, 各个网页访问权限是在装饰器函数 `view.check_login` 中实现的, 只有 `root` 账户有权限访问所有网页, 专户账号只能查看自己策略的网页。

```

15
16 def login(request):
17     # 如果是POST请求, 则说明是点击登录按钮 FORM表单跳转到此的, 那么就要验证密码, 并进行保存session
18     if request.method=="POST":
19         username=request.POST.get('username')
20         password=request.POST.get('password')
21
22         user=User.objects.filter(username=username,password=password)
23
24         if user:
25             """登录成功
26             1. 生成特殊字符串
27             2. 这个字符串当成key, 此key在数据库的session表 (在数据库存一个表名是session的表)
28               中对应一个value
29             3. 在响应中, 用cookies保存这个key, (即向浏览器写一个cookie, 此cookies的值即是这
30               个key特殊字符) """
31             request.session['is_login']='1'
32             # 这个session是用于后面访问每个页面 (即调用每个视图函数时要用到, 即判断是否已经登录, 用此判断)
33             # 这个要存储的session是用于后面, 每个页面上要显示出来, 登录状态的用户名用。
34             request.session['user_id']=user[0].id
35             request.session['user_name']=user[0].username
36             print("{}")
37             return redirect('/real-time/')
38     # 如果是GET请求, 就说明是用户刚开始登录, 使用URL直接进入登录页面的
39     return render(request, HTML_INTEMPLATE_PATH+"login.html")
40     # return render(request, 'login.html')
41

```

```

49 def check_login(f):
50     @wraps(f)
51     def inner(request,*arg,**kwargs):
52         if request.session.get('is_login')=='1':
53
54             login_user = request.session.get('user_name')
55             if login_user == "root":
56                 return f(request,*arg,**kwargs)
57             # 非root用户会强制转换成该专户自己对应的网页，其没有权限访问其他网页
58             else:
59                 web_url = '/' + login_user + '/'
60                 return redirect(web_url)
61         else:
62             return redirect('/login/')
63     return inner
64

```

5.2 实时数据模块

实时数据的左栏：策略总结的分组，是根据 strategy_pnl_total.json 中的 strategy_group 以及 strategy_type groupby 分组生成的 json 数据（详见 view.jsonWriteTotalPnl），仅展示策略管理界面设置为 1 的策略。

```

992
993 SM_TABLE_SHOW = SM_TABLE_ALL.loc[SM_TABLE_ALL['show']=='1']
994 SM_TABLE_SHOW = SM_TABLE_SHOW.sort_values(by=["strategy_class","account_number"])
995 SM_TABLE_SHOW = SM_TABLE_SHOW.reset_index(drop=True)
996 """将设置为展示的策略分发给各个分组"""
997
998 df_group = SM_TABLE_SHOW.groupby(by=["group", "real_or_simulate"])
999 for name, group in df_group:
1000     file_name = '_'.join(name)
1001     generate_group_file(file_name, group)
1002
1003 df_group = SM_TABLE_SHOW.groupby(by=["real_or_simulate","strategy_type"])
1004 for name, group in df_group:
1005     file_name = '_'.join(name)
1006     generate_group_file(file_name, group)

```

5.3 投研分析

此模块是现货价格投研模块，对应 data_path.py 中的现货投研分析数据路径下的数据。

5.4 策略管理

策略管理层级：产品→账号→策略

产品管理：

产品名，净资产，是否 web 显示

账号管理：

产品对应的账号，可一对多，净值清算时会使用该配置表进行产品账号的分类清算，网页前端展示也会使用该表进行中文名匹配。

1. 自动清算：每天 15:30 Django 的定时任务会执行产品净值清算（详情见定时任务管理：task.py productClear）

2. 手动清算：由于交易日中往往会添加很多手动单，所以常用手动清算方法，即每日核算完成之后，双击“手动清算净值.bat”

策略管理:

策略名, 交易日期, 所属账号, 策略分组, 策略类型, 实盘/模拟盘, 是否显示, 策略等级 (产品户为 1, 自营户为 2, 专户为 3, 模拟户为 8, 有助于网页策略表格排序) 以及昨日资产等字段。

出入金: 对应 view.py 中的 `in_out_of_money()`, 网页端提交按钮响应函数为对应的管理函数 `add_strategy`, `add_product`, `add_account`, 并在函数内部调用出入金函数进行后端数据修改。

校准净值与净资产:

由于手续费的差异, 现金管理以及基金托管费, 业绩报酬等等复杂的因素, web 显示很难做到与账户资产的精准跟踪, 所以需要定期手动校准产品与账号的净资产与净值, 以保证网页记录每天净值的准确性。此处只校准净值, 不修改基金份额。

注:

添加或修改修改产品、策略配置请在每天收盘之后进行, 不然第二天将无法上传净值。

5.5 主页

展示所有产品的盈亏比率, 其中各个产品的昨日净资产是通过策略管理中的产品配置表获取的, 且该表的净资产会在定时任务中每日更新。

此外, 也可展示股票实盘分组, 期货期权实盘策略等分组的实时盈亏信息, 分组文件是在 views.py 中 `jsonWriteTotalIPNL` 函数中选择根据字段进行分组生成的, 分组逻辑主要是根据如下三个字段:

策略分组, 策略类型与是否实盘

策略分组: zhang, quant, lifeng, cheng (等专户)

策略类型: stock, future, option

是否实盘: 1, 0

根据如上三个字段可以将所有策略分成类似于 `zhang_real`, `zhang_simulate`, `quant_real`, `real_stock`, `real_option` 等等分组文件, 并选择展示。

5.6 历史净值

根据分组获得分组的名称, 匹配到中文名 (同步加载, 见 JS 脚本及前端代码), get 后端 json 数据, 通过 `highstock` 模板画出历史净值图 (见 `netvalue.js` 脚本详解)

6. JS 脚本及前端代码

6.1 JS 脚本所在路径

../mysite/static/js

6.2 JS 脚本介绍

formatter-monitors.js

保存所有的前端格式函数，诸如底色，红绿，字体大小，加粗等等

使用方法：在 HTML 中引入该脚本，对 BootstrapTable 的样式 data-formmater 进行装饰即可。如：

```
<div id="detail_table_panel" style="...">
  <table id="detail_table"
    data-toggle="table"
    data-toolbar="#toolbar">
    <caption id="detail-table-caption"></caption>
    <thead>
      <tr id = "columnName2">
        <th data-field="stock_code" data-sortable="true">证券代码</th>
        <th data-field="stock_name" data-sortable="true">证券名称</th>
        <th data-field="signal" data-sortable="true">Signal</th>
        <th data-field="long_short" data-sortable="true" data-formmater="duoKongFormater">方向</th>
        <th data-field="market_value" data-sortable="true" >持仓市值</th>
      </tr>
    </thead>
  </table>
</div>
```

addLoadEvent.js

这个脚本实现不停地刷新执行 js 脚本。

```
1 function addLoadEvent(func) {
2   var oldonload = window.onload;
3   if (typeof window.onload != 'function') {
4     window.onload = func;
5   } else {
6     window.onload = function() {
7       oldonload();
8       func();
9     }
10  }
```

main-page.js

主页 PNL 盈亏显示

由于策略名字为“xx_0000”的英文加账号的形式，网页显示策略名字要转换成中文名，这里需要同步加载产品账号配置文件(0product_account_manage.json)，根据账号匹配中文名，前端展示中文名称，其他网页显示策略中文名同理。

```

297 var url_path = "../static/tableData/Manage/product_account_manage.json";
298
299 $.ajaxSettings.async=false;
300 $.getJSON(url_path,function(data) {
301
302     for(var i=0;i<data.length;i++){
303
304         var account_number = data[i].account_number;
305         var django_name = data[i].django_name;
306         django_name_list.push([account_number,django_name]);
307         django_name_dict[account_number] = django_name;
308     }
309 });
310 //console.log(django_name_dict);
311
312
313 function djangoNameFormater(value){
314
315     var new_value = value;
316     var account_number_origin = value.split('_')[1];
317     new_value = value.split('_')[0] + '_' + django_name_dict[account_number_origin] ;
318
319     return '<div >' + new_value + '</div>';
320
321 }
322
323

```

debt-monitor.js

这个控制所有实时数据下面的 HTML 文件的排版布局(Bootstrap), 样式(css)及 Web 数据加载, **setInterval** 设置刷新频率, 点击 Table 弹出其对应的持仓及订单详情, 是用的 `on('click-cell.bs.table', function() 回调函数`, 当前设置持仓每 3 秒一刷新, 其中订单数据量可能会特别大, 所以订单的刷新是通过访问后端数据 **newOrderApi** 获取最新的订单并添加到 Bootstrap 中实现的。

newOrderApi 传入参数为策略名, 客户端当前订单条数, 然后在 debt-monitor 中通过 `addNewOrder()` 函数进行新订单状态的修改或新订单的添加。

订单状态的优先级为:

`priority = {"拒绝":5, "已成":4, "已撤单":3, "部分成交":2, "未成交":1, "OK":0}`

其中, 遍历网页, 只要找到了当前新订单状态的订单 id, 即停止循环遍历, 以减少时间复杂度。

```

85
86 function addNewOrder(url_path){
87
88     $.getJSON(url_path,function (data) {
89
90         $("#order-table-caption").text(data.new_len);
91
92         var priority = {"拒绝":5,"已成":4,"已撤单":3,"部分成交":2,"未成交":1,"OK":0};
93
94         var counts = 0;
95         for (j=0;j<data.data.length;j++){
96
97             var web_data = $("#order_table").bootstrapTable('getData');
98             var web_length = web_data.length;
99             var new_length = data.new_len;
100             var exist_order_ids = [];
101             var new_order_id = data.data[j].system_order_id;
102
103             // 遍历一遍，找到所有的system_order_id,为了减少时间，遍历到当前id即停止
104             var counts = -1;
105             var web_index = -1;
106             for (i=0;i<web_length;i++){
107                 counts += 1;
108                 if (new_order_id == web_data[i].system_order_id){
109                     var web_index = counts;
110                     break;
111                 }
112             }
113             exist_order_ids.push(web_data[i].system_order_id);
114
115             //
116             var web_index = exist_order_ids.indexOf(data.data[j].system_order_id);
117             if (web_index != -1){
118                 var new_status = data.data[j].entrust_status;
119                 var old_status = web_data[web_index].entrust_status;
120
121                 if (priority[new_status] > priority[old_status]){
122                     $('#order_table').bootstrapTable('updateRow', {
123                         index: web_index,
124                         replace:true,
125                         row: data.data[j]
126                     });
127                 }else{
128                     console.log("Not OK");
129                 }
130             }
131             else{
132                 $('#order_table').bootstrapTable('insertRow',
133                 {
134                     index:0,
135                     row:data.data[j]
136                 });
137             }
138         }
139     });
140 }
141
142
143

```

订单 getNewOrder() 函数实现语音播报新订单（订单状态“已成”或“拒绝”）。

```

summaryTable.on('click-cell.bs.table', function(e, column, value, row, $element){
    //alert("Value: " + row);
    // Load detail table according to sector name
    $("#detail-table-caption").text(row.strategy_name);
    //
    $("#strategy_chose").text(row);
    $("#input1").attr("value",$("#detail-table-caption")[0].innerText);
    $("#detail_table_panel").show();
    $("#detail_table").bootstrapTable('refresh',{url: "../static/tableData/" + row.strategy_name + "_position.json"});
}

```

```

65
66   setInterval(function(){
67       $("#detail_table").bootstrapTable('refresh',{silent: true});
68       $("#table_summary").bootstrapTable('refresh',{silent: true});
69       toSummaryData();
70   },3000)
71
72
73
74   setInterval(function(){
75       // $("#order_table").bootstrapTable('refresh',{silent: true});
76       var web_length = $("#order-table-caption")[0].innerText;
77       var strategy_name = $("#detail-table-caption")[0].innerText;
78       var url_path = "../newOrderApi/?strategy_name=" + strategy_name + "&old_len=" + parseInt(web_length);
79       addNewOrder(url_path);
80       //getNewOrder();
81   },6000)
82

```

持仓与订单详情，通过 `jQuery.getJson` 获取后端数据，其中显示样式在 `formatter-monitors.js` 中。

`netvalue-monitors.js`

历史净值网页的 js, 主要是展示各产品及策略的历史净值走势，此处使用了 `highcharts` 与 `hishstock` 模板，大盘基准参数选用上证指数与沪深 300 与创业板指。计算夏普率、最大回撤等函数都在 js 里进行计算。

```

    var json = {};
    json.chart = chart;
    json.title = title;
    //     json.subtitle = subtitle;
    json.credits = credits;
    json.legend = legend;

    json.yAxis = yAxis;
    json.series = series;
    json.plotOptions = plotOptions;
    $('#container').highcharts('stockChart',json);

```

注：先根据上证指数的交易日来对策略净值历史数据进行补全，再选定策略的交易时间段计算大盘指数的净值走势（起点为 1）

7. HTML 详解及注意事项

7.1 HTML 模板所在路径

`../mysite/interface/templates/interface/`

7.2 HTML 介绍

`add_user.html`

响应函数: `view.add_user`，将该用户添加到 `sqlite` 数据库。删除用户的话，需要在双击 `db` 数据库 `interface_user` 表，选中用户，删除即可。

login.html

响应函数: `view.login`, 从数据库中筛选匹配是否有该用户, 如有, 则记录当前登录名与登录状态。logout 则清空浏览器的 session 记录。

main_page.html

主页显示, 后端数据选择股票分组的 pnl

quant_real.html

与 `quant_simulate`, `zhang_real`, `zhang_simulate` 等等属于一样的结构。

research.html

现货价格投研板块, 普通排版

netvalue.html

management.html

后端数据为 `MANAGEMENT_PATH` 下的策略管理文件, 包括产品路径, 产品-账号配对表和策略汇总表。

其中修改策略配置等信息的响应函数在 `view.py` 中的策略管理函数部分, 确认无误提交使用回调函数 `onsubmit()` 进行密码匹配。

注意事项:

Bootstrap 依赖于 jQuery, 所以在引入 bootstrap 之前一定要先引入 jquery.

8. 定时任务管理

任务一: 每天晚上 20:50 分左右将当日所有的 JSON 文件进行备份, 压缩打包存到 `BackUpData` 文件夹下。然后, **初始化策略配置表**, 主要包括, 交易日期更新为明天的交易日, `PNL` 重置为 0, 更新昨日资产到策略配置表等等。初始化产品净资产配置表。

任务二、任务三: 收盘半小时后自动清算策略净值与产品净值 (自动清算未启用, 建议使用盘后手动清算净值)

任务四、任务五: 每天收盘后 15:02 分, 从新浪财经拿到上证指数, 沪深 300 与创业板指的最新收盘价, 作为历史净值的参考指数。

9. POST-Sender

POST 数据的流程如下:

`while True:`

→ 读取各个策略的昨持仓文件

→ 初始化持仓详情

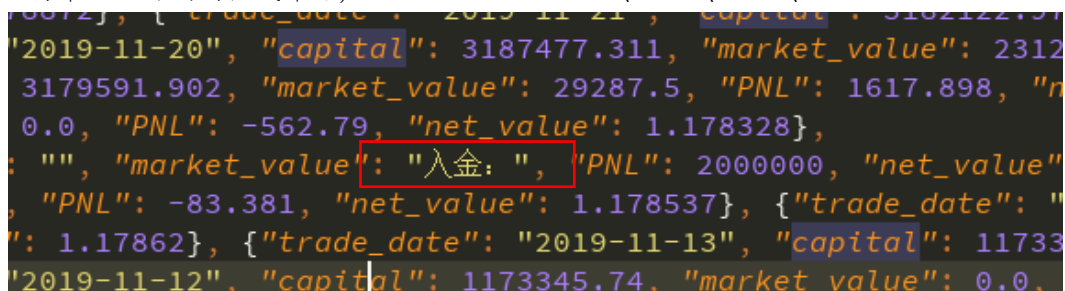
- 记录交易流水最新 index 及订单流水的最新 index，循环读取检查有无新交易发生
 - 若有新交易，则刷新该策略该持仓标的的盈亏，持仓量等信息；
 - 若无新交易，则根据最新价刷新一遍所有的持仓详情
- 发送新持仓
- 发送新 PNL 汇总
- 发送新订单
- 下一轮循环。

注：使用方法参见../PNL_SENDER/pnl_sender5.3/.readme.txt

10. 常见的问题及处理方法

POST 无异常，历史净值未上传

1. 未到净值清算时间，详情查看 task.py 定时任务的启动时间。
2. 按 F12 关闭浏览器缓存 (Disable cache 勾选)，再按 F5 强行刷新。
3. 出入金或修改新策略导致 ts_data.json 或者 nv_detail.json 中的数字变成了字符串。
4. 上证指数或沪深 300 指数未刷新 (历史净值已刷新但是网页显示不出来)。
5. 历史净值文件中包含非编码的中文 (正常应为中文的编码，而不是中文，如下图:)，将中文改为编码格式即可，如：“出入金” → “\u51fa\u5165\u91d1”



```

{"trade_date": "2019-11-21", "capital": 3182122.91,
"2019-11-20", "capital": 3187477.311, "market_value": 2312
3179591.902, "market_value": 29287.5, "PNL": 1617.898, "n
0.0, "PNL": -562.79, "net_value": 1.178328},
: "", "market_value": "入金:", "PNL": 2000000, "net_value"
, "PNL": -83.381, "net_value": 1.178537}, {"trade_date": "
": 1.17862}, {"trade_date": "2019-11-13", "capital": 11733
'2019-11-12", "capital": 1173345.74, "market_value": 0.0,
  
```

6. 策略配置表(strategy_pnl_total.json)出现读写问题。解决办法：用 TempData 中的 strategy_pnl_total.json 或昨日备份文件中的 strategy_pnl_total.json 进行替换即可。

持仓盈亏计算不正确

1. 昨持仓 proclose 或 cost_price 不正确或无文件，或没有最新价。
2. 手动单录入，开平方向，买卖方向录入错误

股票策略出现负持仓数量或与交易软件数量不符

1. 交易流水有问题
2. 手动交易订单未录入

POST SENDER 报错

1. Django 未启动
2. 有人在修改 Django 脚本

3. 数据文件的重要字段变更
4. 其他未知原因，基本都可以重启解决。

策略配置，出入金，校准出现报错:

1. 净值详情_nv_detail.json 最新一条记录的数据格式有出入。
2. 修改信息填错