金融工具函数详细文档

本文档详细说明了三个Python模块中的所有函数，包括输入参数、输出格式、返回值类型和功能描述。

目录

[配置环境 3](#_Toc14125)

[项目配置 5](#_Toc24750)

[测试 5](#_Toc22895)

[global\_tools.py - 金融数据处理和投资组合计算工具包 5](#_Toc32084)

[\_\_init\_\_ 5](#_Toc23901)

[backtesting\_report 7](#_Toc14215)

[cbData\_withdraw 9](#_Toc25064)

[check 11](#_Toc283)

[code\_transfer 12](#_Toc11693)

[df\_to\_sql 14](#_Toc6634)

[etfData\_withdraw 16](#_Toc3590)

[factor\_name 18](#_Toc6949)

[factor\_name\_new 20](#_Toc15264)

[factor\_name\_old 22](#_Toc13853)

[factor\_universe\_withdraw 24](#_Toc22669)

[futureData\_withdraw 26](#_Toc2680)

[indexData\_withdraw 28](#_Toc31798)

[indexFactor\_withdraw 30](#_Toc22254)

[index\_weight\_withdraw 32](#_Toc9117)

[optionData\_withdraw 34](#_Toc8445)

[portfolio\_analyse 36](#_Toc20673)

[process\_future\_code 38](#_Toc4924)

[process\_option\_code 39](#_Toc14892)

[rank\_score\_processing 40](#_Toc3882)

[sql\_to\_timeseries 42](#_Toc13300)

[stockData\_withdraw 44](#_Toc5816)

[stock\_volatility\_calculate 46](#_Toc1459)

[sz\_sh 48](#_Toc4830)

[table\_manager 49](#_Toc15677)

[weight\_df\_datecheck 51](#_Toc26605)

[weight\_df\_standardization 53](#_Toc19494)

[weight\_sum\_check 55](#_Toc3468)

[weight\_sum\_warning 57](#_Toc28245)

[time\_utils.py - 时间处理工具包 59](#_Toc20350)

[Chinese\_valuation\_date 59](#_Toc2602)

[intdate\_transfer 61](#_Toc17820)

[is\_workday 63](#_Toc32533)

[is\_workday\_auto 65](#_Toc17583)

[last\_weeks\_lastday 67](#_Toc15747)

[last\_weeks\_lastday\_df 69](#_Toc22286)

[last\_workday\_auto 71](#_Toc7051)

[last\_workday\_calculate 73](#_Toc807)

[last\_workday\_calculate2 75](#_Toc13259)

[month\_lastday\_df 77](#_Toc5985)

[next\_weeks\_lastday 79](#_Toc18509)

[next\_workday\_auto 81](#_Toc11765)

[next\_workday\_calculate 83](#_Toc12650)

[strdate\_transfer 85](#_Toc28851)

[weeks\_firstday 87](#_Toc24845)

[working\_day\_count 89](#_Toc9168)

[working\_days 91](#_Toc1218)

[working\_days\_list 93](#_Toc30761)

[utils.py - 通用工具包 95](#_Toc25966)

[chunks 95](#_Toc26787)

[close\_all\_connections 97](#_Toc22954)

[contains\_chinese 99](#_Toc17297)

[data\_getting 101](#_Toc2070)

[data\_getting\_glb 103](#_Toc18505)

[data\_reader 105](#_Toc24862)

[file\_withdraw 107](#_Toc16966)

[file\_withdraw2 109](#_Toc13316)

[folder\_creator 111](#_Toc5780)

[folder\_creator2 113](#_Toc8500)

[folder\_creator3 115](#_Toc7508)

[get\_db\_connection 117](#_Toc31780)

[get\_string\_before\_last\_dot 119](#_Toc12437)

[index\_mapping 121](#_Toc338)

[move\_specific\_files 123](#_Toc25309)

[move\_specific\_files2 125](#_Toc10529)

[optiondata\_greeksprocessing 127](#_Toc8983)

[readcsv 128](#_Toc26271)

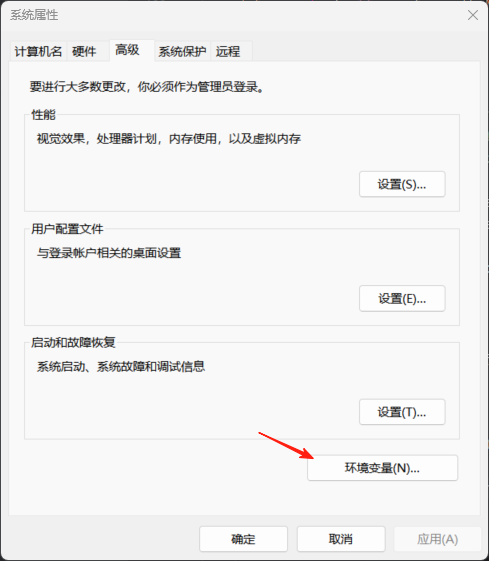
[source\_getting 130](#_Toc188)

[source\_getting2 132](#_Toc22907)

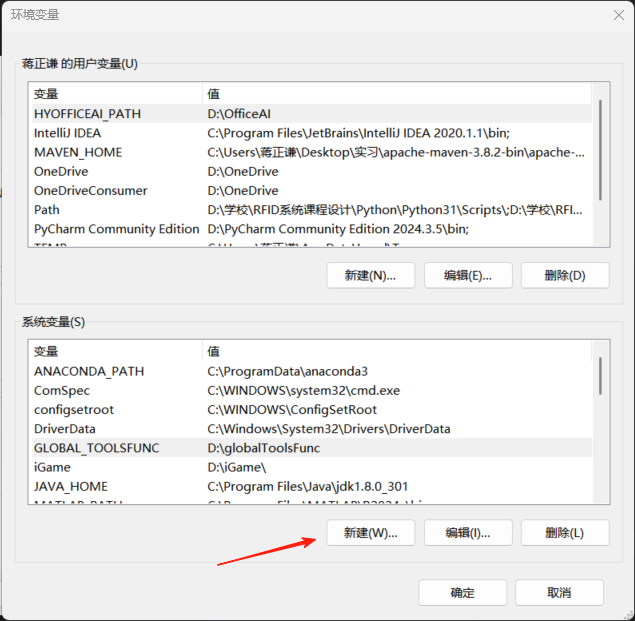
[总结 134](#_Toc6765)

## 配置环境

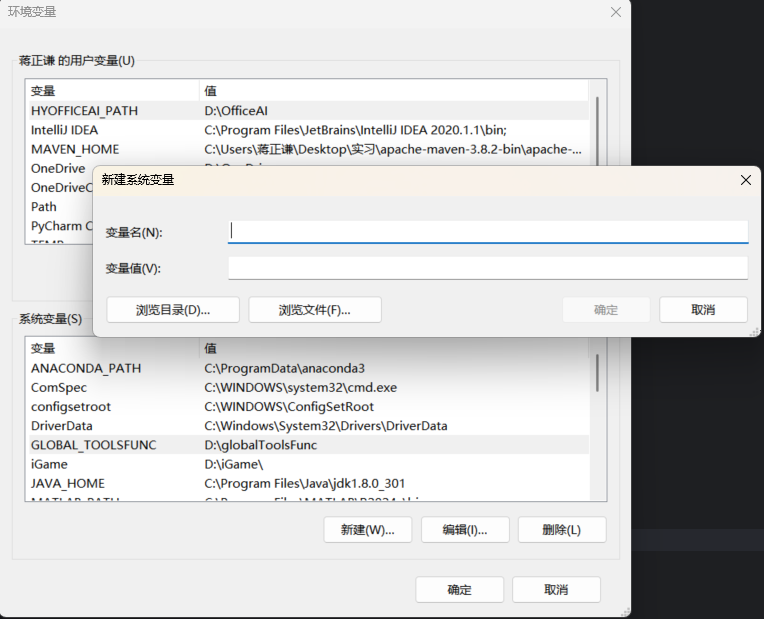
按 Win + R，然后输入： **sysdm.cpl ,3** 打开系统属性



点击环境变量



在系统变量下方点击新建



设置变量

变量名：GLOBAL\_TOOLSFUNC

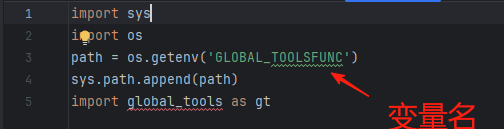
变量值：globalToolsFunc的存放路径（例：D:\globalToolsFunc）

确定之后，一定要依次保存退出

## 项目配置

环境配置成功之后，重新打开编程软件，在项目抬头输入以下代码

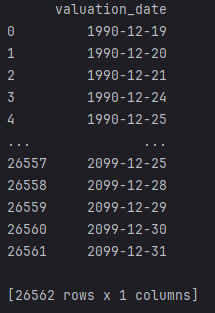
import sys  
import os  
path = os.getenv('GLOBAL\_TOOLSFUNC')  
sys.path.append(path)  
import global\_tools as gt



## 测试

print(gt.Chinese\_valuation\_date())

输入查询交易日代码，若成功返回下图的值代表测试成功



## global\_tools.py - 金融数据处理和投资组合计算工具包

### \_\_init\_\_

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 689 |
| 参数 | config\_path, parameter\_name, delete (默认值: None, None, False) |

#### 功能描述

初始化数据库保存对象  
  
Args:  
 config\_path (str, optional): 配置文件路径  
 parameter\_name (str, optional): 参数名称  
 delete (bool, optional): 是否删除，默认为False

#### 使用示例

```python  
# \_\_init\_\_ 使用示例  
from global\_tools import \_\_init\_\_  
  
# 示例代码  
  
# 调用函数  
result = \_\_init\_\_()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### backtesting\_report

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 649 |
| 参数 | df\_portfolio, outputpath, index\_type, signal\_name (默认值: <ast.Call object at 0x000001D34C31B040>, None, None, 'portfolio') |
| 输入格式 | pandas.DataFrame, str, str, str |
| 输出格式 | None |

#### 功能描述

生成回测报告  
  
对投资组合数据进行回测分析并生成报告  
  
Args:  
 df\_portfolio (pandas.DataFrame, optional): 投资组合数据，默认为空DataFrame  
 outputpath (str, optional): 输出路径，默认为None  
 index\_type (str, optional): 基准指数类型，默认为None  
 signal\_name (str, optional): 信号名称，默认为'portfolio'

#### 详细说明

生成回测报告，包含各种绩效指标

#### 输入参数

df\_portfolio: 投资组合数据, outputpath: 输出路径, index\_type: 指数类型, signal\_name: 信号名称

#### 输出参数

无，生成回测报告

#### 使用示例

```python  
# backtesting\_report 使用示例  
from global\_tools import backtesting\_report  
  
# 示例代码  
# 输入参数: df\_portfolio: 投资组合数据, outputpath: 输出路径, index\_type: 指数类型, signal\_name: 信号名称  
# 输出: 无，生成回测报告  
  
# 调用函数  
result = backtesting\_report()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### cbData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 392 |
| 参数 | start\_date, end\_date, columns, realtime (默认值: None, None, None, False) |
| 输入格式 | str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取可转债数据  
  
获取指定时间范围内的可转债数据  
  
Args:  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: 可转债数据

#### 详细说明

提取可转债数据，支持日频数据

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

可转债数据

#### 使用示例

```python  
# cbData\_withdraw 使用示例  
from global\_tools import cbData\_withdraw  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: 可转债数据  
  
# 调用函数  
result = cbData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### check

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 572 |
| 参数 | df |

#### 使用示例

```python  
# check 使用示例  
from global\_tools import check  
  
# 示例代码  
  
# 调用函数  
result = check()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### code\_transfer

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 86 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

股票代码格式转换  
  
将数字格式的股票代码转换为标准格式（如000001.SZ）  
  
Args:  
 df (pandas.DataFrame): 包含code列的数据框  
  
Returns:  
 pandas.DataFrame: 处理后的数据框，code列转换为标准格式

#### 详细说明

股票代码格式转换，将数字代码转换为带交易所后缀的标准格式

#### 输入参数

df: 包含code列的数据框

#### 输出参数

code列格式化为标准格式的数据框

#### 使用示例

```python  
# code\_transfer 使用示例  
from global\_tools import code\_transfer  
  
# 示例代码  
# 输入参数: df: 包含code列的数据框  
# 输出: code列格式化为标准格式的数据框  
  
# 调用函数  
result = code\_transfer()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### df\_to\_sql

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 711 |
| 参数 | df, delete\_name, delet\_key (默认值: <ast.Call object at 0x000001D34C3262B0>, None, None) |

#### 功能描述

调用此函数之前需要先初始化class，sqlSaving\_main(config\_path,parameter\_name,delete)

Config\_path为入库标准化配置文件（yaml）的路径，parameter\_name为要入库的part的参数，delete为是否要删除以前的文件，入库的配置文件格式如下：

文本

AI 生成的内容可能不正确。

optionfuture\_holding\_his:  
 table\_name: "history\_futureoptionholding"  
 db\_url: ""  
 chunk\_size: 20000  
 workers: 4  
 private\_keys:  
 - "product\_code"  
 - 'code'  
 schema:  
 valuation\_date: { type: String, length: 50 }  
 simulation: { type: String, length: 50 }  
 product\_code: { type: String, length: 50 }  
 code: { type: String, length: 50 }  
 direction: { type: String, length: 50 }  
 HoldingQty: { type: Float }  
 delta: { type: Float }  
 mkt\_value: { type: Float }  
 daily\_profit: { type: Float }  
 update\_time: { type: String, length: 50 }

将DataFrame保存到数据库  
  
Args:  
 df (pandas.DataFrame, optional): 要保存的数据框，默认为空DataFrame  
 delete\_name (str, optional): 删除名称，默认为None  
 delet\_key (str, optional): 删除键，默认为None

#### 使用示例

```python  
# df\_to\_sql 使用示例  
from global\_tools import df\_to\_sql  
  
# 示例代码  
  
# 调用函数

Sm=sqlSaving\_main(config\_path,parameter\_name,delete)  
result = sm.df\_to\_sql()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### etfData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 363 |
| 参数 | start\_date, end\_date, columns, realtime (默认值: None, None, None, False) |
| 输入格式 | str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取ETF数据  
  
获取指定时间范围内的ETF数据  
  
Args:  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: ETF数据

#### 详细说明

提取ETF数据，支持日频和实时数据

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

ETF数据

#### 使用示例

```python  
# etfData\_withdraw 使用示例  
from global\_tools import etfData\_withdraw  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: ETF数据  
  
# 调用函数  
result = etfData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### factor\_name

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 158 |
| 参数 | inputpath\_factor |
| 输入格式 | str |
| 输出格式 | tuple |

#### 功能描述

从因子文件中提取因子名称  
  
从MATLAB格式的因子文件中提取Barra因子和行业因子名称  
  
Args:  
 inputpath\_factor (str): 因子文件路径  
  
Returns:  
 tuple: (barra\_name, industry\_name) - Barra因子名称列表和行业因子名称列表

#### 详细说明

从因子文件中提取因子名称，区分Barra因子和行业因子

#### 输入参数

inputpath\_factor: 因子文件路径

#### 输出参数

(barra\_name, industry\_name): 从文件中提取的因子名称列表

#### 使用示例

```python  
# factor\_name 使用示例  
from global\_tools import factor\_name  
  
# 示例代码  
# 输入参数: inputpath\_factor: 因子文件路径  
# 输出: (barra\_name, industry\_name): 从文件中提取的因子名称列表  
  
# 调用函数  
result = factor\_name()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### factor\_name\_new

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 140 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | tuple |

#### 功能描述

获取新版因子名称  
  
返回新版本的Barra因子名称和行业因子名称列表  
  
Returns:  
 tuple: (barra\_name, industry\_name) - Barra因子名称列表和行业因子名称列表

#### 详细说明

获取新版因子名称，包括Barra因子和行业因子

#### 输入参数

无

#### 输出参数

(barra\_name, industry\_name): 新版因子名称列表

#### 使用示例

```python  
# factor\_name\_new 使用示例  
from global\_tools import factor\_name\_new  
  
# 示例代码  
# 输出: (barra\_name, industry\_name): 新版因子名称列表  
  
# 调用函数  
result = factor\_name\_new()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### factor\_name\_old

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 122 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | tuple |

#### 功能描述

获取旧版因子名称  
  
返回旧版本的Barra因子名称和行业因子名称列表  
  
Returns:  
 tuple: (barra\_name, industry\_name) - Barra因子名称列表和行业因子名称列表

#### 详细说明

获取旧版因子名称，包括Barra因子和行业因子

#### 输入参数

无

#### 输出参数

(barra\_name, industry\_name): 旧版因子名称列表

#### 使用示例

```python  
# factor\_name\_old 使用示例  
from global\_tools import factor\_name\_old  
  
# 示例代码  
# 输出: (barra\_name, industry\_name): 旧版因子名称列表  
  
# 调用函数  
result = factor\_name\_old()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### factor\_universe\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 225 |
| 参数 | type (默认值: 'new') |
| 输入格式 | str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

获取股票池数据  
  
根据类型获取新版本或旧版本的股票池数据  
  
Args:  
 type (str, optional): 类型（'new'或'old'），默认为'new'  
  
Returns:  
 pandas.DataFrame: 股票池数据

#### 详细说明

获取股票池数据，支持新旧两种类型

#### 输入参数

type: 类型（new或old），默认为new

#### 输出参数

股票池数据

#### 使用示例

```python  
# factor\_universe\_withdraw 使用示例  
from global\_tools import factor\_universe\_withdraw  
  
# 示例代码  
# 输入参数: type: 类型（new或old），默认为new  
# 输出: 股票池数据  
  
# 调用函数  
result = factor\_universe\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### futureData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 452 |
| 参数 | start\_date, end\_date, columns, realtime (默认值: None, None, None, False) |
| 输入格式 | str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取期货数据  
  
获取指定时间范围内的期货数据  
  
Args:  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: 期货数据

#### 详细说明

提取期货数据，支持日频和实时数据

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

期货数据

#### 使用示例

```python  
# futureData\_withdraw 使用示例  
from global\_tools import futureData\_withdraw  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: 期货数据  
  
# 调用函数  
result = futureData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### indexData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 279 |
| 参数 | index\_type, start\_date, end\_date, columns, realtime (默认值: None, None, None, None, False) |
| 输入格式 | str, str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取指数收益率数据  
  
获取指定指数在指定时间范围内的数据  
  
Args:  
 index\_type (str, optional): 指数类型  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: 指数数据

#### 详细说明

提取指数收益率数据，支持日频和实时数据

#### 输入参数

index\_type: 指数类型, start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

指数收益率数据

#### 使用示例

```python  
# indexData\_withdraw 使用示例  
from global\_tools import indexData\_withdraw  
  
# 示例代码  
# 输入参数: index\_type: 指数类型, start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: 指数收益率数据  
  
# 调用函数  
result = indexData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### indexFactor\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 310 |
| 参数 | index\_type, start\_date, end\_date (默认值: None, None) |
| 输入格式 | str, str, str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取指数因子暴露数据  
  
获取指定指数在指定时间范围内的因子暴露数据  
  
Args:  
 index\_type (str): 指数类型  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
  
Returns:  
 pandas.DataFrame: 因子暴露数据

#### 详细说明

提取指数因子暴露数据

#### 输入参数

index\_type: 指数类型, start\_date: 开始日期, end\_date: 结束日期

#### 输出参数

因子暴露数据

#### 使用示例

```python  
# indexFactor\_withdraw 使用示例  
from global\_tools import indexFactor\_withdraw  
  
# 示例代码  
# 输入参数: index\_type: 指数类型, start\_date: 开始日期, end\_date: 结束日期  
# 输出: 因子暴露数据  
  
# 调用函数  
result = indexFactor\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### index\_weight\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 249 |
| 参数 | index\_type, available\_date |
| 输入格式 | str, str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取指数权重股数据  
  
获取指定指数在指定日期的成分股权重数据  
  
Args:  
 index\_type (str): 指数类型  
 available\_date (str): 日期  
  
Returns:  
 pandas.DataFrame: 权重股数据，包含code和weight列

#### 详细说明

提取指数权重股数据，支持本地和SQL数据源

#### 输入参数

index\_type: 指数类型, available\_date: 日期

#### 输出参数

权重股数据，包含code和weight列

#### 使用示例

```python  
# index\_weight\_withdraw 使用示例  
from global\_tools import index\_weight\_withdraw  
  
# 示例代码  
# 输入参数: index\_type: 指数类型, available\_date: 日期  
# 输出: 权重股数据，包含code和weight列  
  
# 调用函数  
result = index\_weight\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### optionData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 423 |
| 参数 | start\_date, end\_date, columns, realtime (默认值: None, None, None, False) |
| 输入格式 | str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取期权数据  
  
获取指定时间范围内的期权数据  
  
Args:  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: 期权数据

#### 详细说明

提取期权数据，支持日频和实时数据

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

期权数据

#### 使用示例

```python  
# optionData\_withdraw 使用示例  
from global\_tools import optionData\_withdraw  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: 期权数据  
  
# 调用函数  
result = optionData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### portfolio\_analyse

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 589 |
| 参数 | df\_holding, account\_money, cost\_stock, cost\_etf, cost\_future, cost\_option, cost\_convertiblebond, realtime, weight\_standardize (默认值: <ast.Call object at 0x000001D34C303F10>, 10000000, 0.00085, 0.0003, 6e-05, 0.01, 0.0007, False, True) |
| 输入格式 | pandas.DataFrame, float, float, float, float, float, float, bool, bool |
| 输出格式 | tuple |

#### 功能描述

投资组合分析主函数  
  
对投资组合进行完整的收益分析，包括收益计算、风险分析等  
  
Args:  
 df\_holding (pandas.DataFrame, optional): 持仓数据，默认为空DataFrame  
 account\_money (float, optional): 账户资金，默认为10000000  
 cost\_stock (float, optional): 股票交易成本，默认为0.00085  
 cost\_etf (float, optional): ETF交易成本，默认为0.0003  
 cost\_future (float, optional): 期货交易成本，默认为0.00006  
 cost\_option (float, optional): 期权交易成本，默认为0.01  
 cost\_convertiblebond (float, optional): 可转债交易成本，默认为0.0007  
 realtime (bool, optional): 是否为实时数据，默认为False  
 weight\_standardize (bool, optional): 是否标准化权重，默认为True  
  
Returns:  
 tuple: (df\_info, df\_detail) - 投资组合汇总信息和详细数据

#### 详细说明

投资组合分析，计算收益、风险等指标

#### 输入参数

df\_holding: 持仓数据, account\_money: 账户资金, cost\_stock: 股票成本, cost\_etf: ETF成本, cost\_future: 期货成本, cost\_option: 期权成本, cost\_convertiblebond: 可转债成本, realtime: 是否实时数据, weight\_standardize: 是否标准化权重

df\_holding的格式要求必须有valuation\_date和code,如果有quantity则account\_money可以不填写，如果有weight，account\_money默认为10000万。如果需要计算超额则index\_type必须填写，默认index\_return为0. 如果要计算多portfolio的多天，则必须要求有portfolio\_name列，如果单一组合则不需要portfolio\_name列。

#### 输出参数

(df\_info, df\_detail): 投资组合分析结果

#### 使用示例

```python  
# portfolio\_analyse 使用示例  
from global\_tools import portfolio\_analyse  
  
# 示例代码  
# 输入参数: df\_holding: 持仓数据, account\_money: 账户资金, cost\_stock: 股票成本, cost\_etf: ETF成本, cost\_future: 期货成本, cost\_option: 期权成本, cost\_convertiblebond: 可转债成本, realtime: 是否实时数据, weight\_standardize: 是否标准化权重  
# 输出: (df\_info, df\_detail): 投资组合分析结果  
  
# 调用函数  
result = portfolio\_analyse()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### process\_future\_code

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 525 |
| 参数 | code |

#### 使用示例

```python  
# process\_future\_code 使用示例  
from global\_tools import process\_future\_code  
  
# 示例代码  
  
# 调用函数  
result = process\_future\_code()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### process\_option\_code

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 541 |
| 参数 | code |

#### 使用示例

```python  
# process\_option\_code 使用示例  
from global\_tools import process\_option\_code  
  
# 示例代码  
  
# 调用函数  
result = process\_option\_code()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### rank\_score\_processing

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 56 |
| 参数 | df\_score |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

标准化分数生成  
  
对每个日期的分数进行标准化处理，生成标准正态分布的分数  
  
Args:  
 df\_score (pandas.DataFrame): 包含valuation\_date和code列的数据框  
  
Returns:  
 pandas.DataFrame: 处理后的数据框，包含标准化后的final\_score列

#### 详细说明

标准化分数生成，对每个日期的股票进行排名并标准化处理

#### 输入参数

df\_score: 包含valuation\_date和code列的数据框

#### 输出参数

包含final\_score列的处理后数据框

#### 使用示例

```python  
# rank\_score\_processing 使用示例  
from global\_tools import rank\_score\_processing  
  
# 示例代码  
# 输入参数: df\_score: 包含valuation\_date和code列的数据框  
# 输出: 包含final\_score列的处理后数据框  
  
# 调用函数  
result = rank\_score\_processing()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### sql\_to\_timeseries

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 35 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

将SQL查询结果转换为时间序列格式  
  
将包含valuation\_date、code、value三列的DataFrame转换为宽格式时间序列  
  
Args:  
 df (pandas.DataFrame): 包含valuation\_date、code、value列的数据框  
  
Returns:  
 pandas.DataFrame: 转换后的时间序列数据框，每行一个日期，每列一个代码

#### 详细说明

将SQL查询结果转换为时间序列格式，处理NULL值并进行数据透视

#### 输入参数

df: 包含valuation\_date, code, value列的数据框

#### 输出参数

处理后的时间序列数据框，以valuation\_date为索引，code为列名

#### 使用示例

```python  
# sql\_to\_timeseries 使用示例  
from global\_tools import sql\_to\_timeseries  
  
# 示例代码  
# 输入参数: df: 包含valuation\_date, code, value列的数据框  
# 输出: 处理后的时间序列数据框，以valuation\_date为索引，code为列名  
  
# 调用函数  
result = sql\_to\_timeseries()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### stockData\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 333 |
| 参数 | start\_date, end\_date, columns, realtime (默认值: None, None, None, False) |
| 输入格式 | str, str, list, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取股票数据  
  
获取指定时间范围内的股票数据  
  
Args:  
 start\_date (str, optional): 开始日期  
 end\_date (str, optional): 结束日期  
 columns (list, optional): 需要的列名  
 realtime (bool, optional): 是否为实时数据，默认为False  
  
Returns:  
 pandas.DataFrame: 股票数据

#### 详细说明

提取股票数据，支持日频和实时数据

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据

#### 输出参数

股票数据

#### 使用示例

```python  
# stockData\_withdraw 使用示例  
from global\_tools import stockData\_withdraw  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期, columns: 列名列表, realtime: 是否实时数据  
# 输出: 股票数据  
  
# 调用函数  
result = stockData\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### stock\_volatility\_calculate

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 207 |
| 参数 | df, available\_date |
| 输入格式 | pandas.DataFrame, str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

计算股票波动率  
  
计算指定日期之前的股票收益率波动率（滚动252天标准差）  
  
Args:  
 df (pandas.DataFrame): 包含valuation\_date列的数据框  
 available\_date (str): 计算截止日期  
  
Returns:  
 pandas.DataFrame: 计算后的波动率数据框

#### 详细说明

计算股票波动率，使用248天滚动窗口计算标准差

#### 输入参数

df: 包含valuation\_date列的数据框, available\_date: 日期

#### 输出参数

计算后的波动率数据框

#### 使用示例

```python  
# stock\_volatility\_calculate 使用示例  
from global\_tools import stock\_volatility\_calculate  
  
# 示例代码  
# 输入参数: df: 包含valuation\_date列的数据框, available\_date: 日期  
# 输出: 计算后的波动率数据框  
  
# 调用函数  
result = stock\_volatility\_calculate()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### sz\_sh

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 102 |
| 参数 | x |

#### 使用示例

```python  
# sz\_sh 使用示例  
from global\_tools import sz\_sh  
  
# 示例代码  
  
# 调用函数  
result = sz\_sh()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### table\_manager

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 726 |
| 参数 | config\_path, table\_name |
| 输入格式 | str, str |
| 输出格式 | bool |

#### 功能描述

删除指定的数据库表  
  
删除数据库中指定的表  
  
Args:  
 config\_path (str): 配置文件路径  
 table\_name (str): 要删除的表名  
  
Returns:  
 bool: 操作是否成功

#### 详细说明

删除指定的数据库表

#### 输入参数

config\_path: 配置文件路径, table\_name: 表名

#### 输出参数

操作是否成功

#### 使用示例

```python  
# table\_manager 使用示例  
from global\_tools import table\_manager  
  
# 示例代码  
# 输入参数: config\_path: 配置文件路径, table\_name: 表名  
# 输出: 操作是否成功  
  
# 调用函数  
result = table\_manager()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### weight\_df\_datecheck

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 563 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | None |

#### 功能描述

检查权重数据的日期完整性  
  
检查DataFrame中的日期是否连续，如果不连续则抛出异常  
  
Args:  
 df (pandas.DataFrame): 包含valuation\_date列的数据框

#### 详细说明

检查权重数据框的日期连续性，确保没有缺失的交易日

#### 输入参数

df: 包含valuation\_date列的数据框

#### 输出参数

无，检查日期连续性

#### 使用示例

```python  
# weight\_df\_datecheck 使用示例  
from global\_tools import weight\_df\_datecheck  
  
# 示例代码  
# 输入参数: df: 包含valuation\_date列的数据框  
# 输出: 无，检查日期连续性  
  
# 调用函数  
result = weight\_df\_datecheck()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### weight\_df\_standardization

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 481 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

标准化权重数据  
  
对包含code列的DataFrame进行代码标准化处理，支持股票、期货、期权等不同类型  
  
Args:  
 df (pandas.DataFrame): 包含code列的数据框  
  
Returns:  
 pandas.DataFrame: 标准化后的数据框

#### 详细说明

标准化权重数据，处理股票、期货、期权代码格式

#### 输入参数

df: 包含code列的数据框

#### 输出参数

标准化后的数据框

#### 使用示例

```python  
# weight\_df\_standardization 使用示例  
from global\_tools import weight\_df\_standardization  
  
# 示例代码  
# 输入参数: df: 包含code列的数据框  
# 输出: 标准化后的数据框  
  
# 调用函数  
result = weight\_df\_standardization()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### weight\_sum\_check

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 177 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

检查权重和  
  
检查权重列的和是否为1，如果不是则进行标准化  
  
Args:  
 df (pandas.DataFrame): 包含weight列的数据框  
  
Returns:  
 pandas.DataFrame: 处理后的数据框，权重已标准化

#### 详细说明

检查权重和，如果小于0.99则进行标准化处理

#### 输入参数

df: 包含weight列的数据框

#### 输出参数

权重和标准化后的数据框

#### 使用示例

```python  
# weight\_sum\_check 使用示例  
from global\_tools import weight\_sum\_check  
  
# 示例代码  
# 输入参数: df: 包含weight列的数据框  
# 输出: 权重和标准化后的数据框  
  
# 调用函数  
result = weight\_sum\_check()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### weight\_sum\_warning

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | global\_tools.py |
| 定义行号 | 194 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | None |

#### 功能描述

权重和警告  
  
检查权重列的和是否在合理范围内，如果不在则发出警告  
  
Args:  
 df (pandas.DataFrame): 包含weight列的数据框

#### 详细说明

权重和警告，检查权重和是否在合理范围内

#### 输入参数

df: 包含weight列的数据框

#### 输出参数

无，仅打印警告信息

#### 使用示例

```python  
# weight\_sum\_warning 使用示例  
from global\_tools import weight\_sum\_warning  
  
# 示例代码  
# 输入参数: df: 包含weight列的数据框  
# 输出: 无，仅打印警告信息  
  
# 调用函数  
result = weight\_sum\_warning()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

## time\_utils.py - 时间处理工具包

### Chinese\_valuation\_date

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 17 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | pandas.DataFrame |

#### 功能描述

获取中国交易日期数据  
  
从配置的数据源获取中国股市交易日期列表  
  
Returns:  
 pandas.DataFrame: 包含valuation\_date列的交易日期数据框

#### 详细说明

获取中国交易日期数据

#### 输入参数

无

#### 输出参数

交易日期数据

#### 使用示例

```python  
# Chinese\_valuation\_date 使用示例  
from time\_utils import Chinese\_valuation\_date  
  
# 示例代码  
# 输出: 交易日期数据  
  
# 调用函数  
result = Chinese\_valuation\_date()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### intdate\_transfer

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 234 |
| 参数 | date |
| 输入格式 | str/datetime |
| 输出格式 | str |

#### 功能描述

日期转整数格式  
  
将日期转换为YYYYMMDD格式的整数字符串  
  
Args:  
 date (str/datetime): 日期  
  
Returns:  
 str: 整数格式日期字符串（YYYYMMDD）

#### 详细说明

日期转整数格式

#### 输入参数

date: 日期

#### 输出参数

整数格式日期

#### 使用示例

```python  
# intdate\_transfer 使用示例  
from time\_utils import intdate\_transfer  
  
# 示例代码  
# 输入参数: date: 日期  
# 输出: 整数格式日期  
  
# 调用函数  
result = intdate\_transfer()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### is\_workday

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 174 |
| 参数 | today |
| 输入格式 | str |
| 输出格式 | bool |

#### 功能描述

判断是否为工作日  
  
检查指定日期是否为交易日  
  
Args:  
 today (str): 日期字符串  
  
Returns:  
 bool: 是否为工作日

#### 详细说明

判断是否为工作日

#### 输入参数

today: 日期

#### 输出参数

是否为工作日

#### 使用示例

```python  
# is\_workday 使用示例  
from time\_utils import is\_workday  
  
# 示例代码  
# 输入参数: today: 日期  
# 输出: 是否为工作日  
  
# 调用函数  
result = is\_workday()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### is\_workday\_auto

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 213 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | bool |

#### 功能描述

判断今天是否为工作日  
  
自动检查当前日期是否为交易日  
  
Returns:  
 bool: 是否为工作日

#### 详细说明

判断今天是否为工作日

#### 输入参数

无

#### 输出参数

是否为工作日

#### 使用示例

```python  
# is\_workday\_auto 使用示例  
from time\_utils import is\_workday\_auto  
  
# 示例代码  
# 输出: 是否为工作日  
  
# 调用函数  
result = is\_workday\_auto()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### last\_weeks\_lastday

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 350 |
| 参数 | date |
| 输入格式 | str |
| 输出格式 | str |

#### 功能描述

获取指定日期的上周最后工作日  
  
根据指定日期获取其上周的最后一个交易日  
  
Args:  
 date (str): 指定日期  
  
Returns:  
 str: 上周最后工作日

#### 详细说明

获取指定日期的上周最后工作日

#### 输入参数

date: 指定日期

#### 输出参数

上周最后工作日

#### 使用示例

```python  
# last\_weeks\_lastday 使用示例  
from time\_utils import last\_weeks\_lastday  
  
# 示例代码  
# 输入参数: date: 指定日期  
# 输出: 上周最后工作日  
  
# 调用函数  
result = last\_weeks\_lastday()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### last\_weeks\_lastday\_df

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 328 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | str |

#### 功能描述

获取上周最后工作日  
  
获取上周的最后一个交易日  
  
Returns:  
 str: 上周最后工作日

#### 详细说明

获取上周最后工作日

#### 输入参数

无

#### 输出参数

上周最后工作日

#### 使用示例

```python  
# last\_weeks\_lastday\_df 使用示例  
from time\_utils import last\_weeks\_lastday\_df  
  
# 示例代码  
# 输出: 上周最后工作日  
  
# 调用函数  
result = last\_weeks\_lastday\_df()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### last\_workday\_auto

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 70 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | str |

#### 功能描述

获取上一个工作日  
  
自动获取当前日期的上一个交易日  
  
Returns:  
 str: 上一个工作日的日期字符串

#### 详细说明

获取上一个工作日

#### 输入参数

无

#### 输出参数

上一个工作日

#### 使用示例

```python  
# last\_workday\_auto 使用示例  
from time\_utils import last\_workday\_auto  
  
# 示例代码  
# 输出: 上一个工作日  
  
# 调用函数  
result = last\_workday\_auto()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### last\_workday\_calculate

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 95 |
| 参数 | x |
| 输入格式 | str/datetime |
| 输出格式 | str |

#### 功能描述

计算指定日期的上一个工作日  
  
根据给定的日期计算其上一个交易日  
  
Args:  
 x (str/datetime): 指定日期  
  
Returns:  
 str: 上一个工作日的日期字符串

#### 详细说明

计算指定日期的上一个工作日

#### 输入参数

x: 指定日期

#### 输出参数

上一个工作日

#### 使用示例

```python  
# last\_workday\_calculate 使用示例  
from time\_utils import last\_workday\_calculate  
  
# 示例代码  
# 输入参数: x: 指定日期  
# 输出: 上一个工作日  
  
# 调用函数  
result = last\_workday\_calculate()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### last\_workday\_calculate2

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 149 |
| 参数 | df\_score |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

批量计算上一个工作日  
  
对DataFrame中的每个日期计算其上一个交易日  
  
Args:  
 df\_score (pandas.DataFrame): 包含date列的数据框  
  
Returns:  
 pandas.DataFrame: 处理后的数据框，date列更新为上一个工作日

#### 详细说明

批量计算上一个工作日

#### 输入参数

df\_score: 包含date列的数据框

#### 输出参数

处理后的数据框

#### 使用示例

```python  
# last\_workday\_calculate2 使用示例  
from time\_utils import last\_workday\_calculate2  
  
# 示例代码  
# 输入参数: df\_score: 包含date列的数据框  
# 输出: 处理后的数据框  
  
# 调用函数  
result = last\_workday\_calculate2()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### month\_lastday\_df

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 314 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | list |

#### 功能描述

获取每月最后工作日  
  
获取每个月的最后一个交易日  
  
Returns:  
 list: 每月最后工作日列表

#### 详细说明

获取每月最后工作日

#### 输入参数

无

#### 输出参数

每月最后工作日列表

#### 使用示例

```python  
# month\_lastday\_df 使用示例  
from time\_utils import month\_lastday\_df  
  
# 示例代码  
# 输出: 每月最后工作日列表  
  
# 调用函数  
result = month\_lastday\_df()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### next\_weeks\_lastday

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 398 |
| 参数 | date |
| 输入格式 | str |
| 输出格式 | str |

#### 功能描述

获取下周最后工作日  
  
根据指定日期获取其下周的最后一个交易日  
  
Args:  
 date (str): 日期  
  
Returns:  
 str: 下周最后工作日

#### 详细说明

获取下周最后工作日

#### 输入参数

date: 日期

#### 输出参数

下周最后工作日

#### 使用示例

```python  
# next\_weeks\_lastday 使用示例  
from time\_utils import next\_weeks\_lastday  
  
# 示例代码  
# 输入参数: date: 日期  
# 输出: 下周最后工作日  
  
# 调用函数  
result = next\_weeks\_lastday()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### next\_workday\_auto

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 45 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | str |

#### 功能描述

获取下一个工作日  
  
自动获取当前日期的下一个交易日  
  
Returns:  
 str: 下一个工作日的日期字符串

#### 详细说明

获取下一个工作日

#### 输入参数

无

#### 输出参数

下一个工作日

#### 使用示例

```python  
# next\_workday\_auto 使用示例  
from time\_utils import next\_workday\_auto  
  
# 示例代码  
# 输出: 下一个工作日  
  
# 调用函数  
result = next\_workday\_auto()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### next\_workday\_calculate

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 119 |
| 参数 | x |
| 输入格式 | str/datetime |
| 输出格式 | str |

#### 功能描述

计算指定日期的下一个工作日  
  
根据给定的日期计算其下一个交易日  
  
Args:  
 x (str/datetime): 指定日期  
  
Returns:  
 str: 下一个工作日的日期字符串

#### 详细说明

计算指定日期的下一个工作日

#### 输入参数

x: 指定日期

#### 输出参数

下一个工作日

#### 使用示例

```python  
# next\_workday\_calculate 使用示例  
from time\_utils import next\_workday\_calculate  
  
# 示例代码  
# 输入参数: x: 指定日期  
# 输出: 下一个工作日  
  
# 调用函数  
result = next\_workday\_calculate()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### strdate\_transfer

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 251 |
| 参数 | date |
| 输入格式 | str/datetime |
| 输出格式 | str |

#### 功能描述

日期转字符串格式  
  
将日期转换为YYYY-MM-DD格式的字符串  
  
Args:  
 date (str/datetime): 日期  
  
Returns:  
 str: 字符串格式日期（YYYY-MM-DD）

#### 详细说明

日期转字符串格式

#### 输入参数

date: 日期

#### 输出参数

字符串格式日期

#### 使用示例

```python  
# strdate\_transfer 使用示例  
from time\_utils import strdate\_transfer  
  
# 示例代码  
# 输入参数: date: 日期  
# 输出: 字符串格式日期  
  
# 调用函数  
result = strdate\_transfer()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### weeks\_firstday

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 375 |
| 参数 | date |
| 输入格式 | str |
| 输出格式 | str |

#### 功能描述

获取周第一个工作日  
  
根据指定日期获取其所在周的第一个交易日  
  
Args:  
 date (str): 日期  
  
Returns:  
 str: 周第一个工作日

#### 详细说明

获取周第一个工作日

#### 输入参数

date: 日期

#### 输出参数

周第一个工作日

#### 使用示例

```python  
# weeks\_firstday 使用示例  
from time\_utils import weeks\_firstday  
  
# 示例代码  
# 输入参数: date: 日期  
# 输出: 周第一个工作日  
  
# 调用函数  
result = weeks\_firstday()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### working\_day\_count

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 294 |
| 参数 | start\_date, end\_date |
| 输入格式 | str, str |
| 输出格式 | int |

#### 功能描述

计算工作日天数  
  
计算指定日期范围内的交易日数量  
  
Args:  
 start\_date (str): 开始日期  
 end\_date (str): 结束日期  
  
Returns:  
 int: 工作日天数

#### 详细说明

计算工作日天数

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期

#### 输出参数

工作日天数

#### 使用示例

```python  
# working\_day\_count 使用示例  
from time\_utils import working\_day\_count  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期  
# 输出: 工作日天数  
  
# 调用函数  
result = working\_day\_count()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### working\_days

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 196 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

筛选工作日数据  
  
从DataFrame中筛选出只在交易日的数据  
  
Args:  
 df (pandas.DataFrame): 包含date列的数据框  
  
Returns:  
 pandas.DataFrame: 只包含交易日数据的DataFrame

#### 详细说明

筛选工作日数据

#### 输入参数

df: 包含date列的数据框

#### 输出参数

处理后的数据框

#### 使用示例

```python  
# working\_days 使用示例  
from time\_utils import working\_days  
  
# 示例代码  
# 输入参数: df: 包含date列的数据框  
# 输出: 处理后的数据框  
  
# 调用函数  
result = working\_days()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### working\_days\_list

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | time\_utils.py |
| 定义行号 | 268 |
| 参数 | start\_date, end\_date |
| 输入格式 | str, str |
| 输出格式 | list |

#### 功能描述

获取工作日列表  
  
获取指定日期范围内的所有交易日列表  
  
Args:  
 start\_date (str): 开始日期  
 end\_date (str): 结束日期  
  
Returns:  
 list: 工作日日期列表

#### 详细说明

获取工作日列表

#### 输入参数

start\_date: 开始日期, end\_date: 结束日期

#### 输出参数

工作日列表

#### 使用示例

```python  
# working\_days\_list 使用示例  
from time\_utils import working\_days\_list  
  
# 示例代码  
# 输入参数: start\_date: 开始日期, end\_date: 结束日期  
# 输出: 工作日列表  
  
# 调用函数  
result = working\_days\_list()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

## utils.py - 通用工具包

### chunks

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 481 |
| 参数 | lst, n |
| 输入格式 | list, int |
| 输出格式 | list |

#### 功能描述

等分列表  
  
将列表等分为n个子列表  
  
Args:  
 lst (list): 要等分的列表  
 n (int): 等分数量  
  
Returns:  
 list: 等分后的列表

#### 详细说明

等分列表

#### 输入参数

lst: 要等分的列表, n: 等分数量

#### 输出参数

等分后的列表

#### 使用示例

```python  
# chunks 使用示例  
from utils import chunks  
  
# 示例代码  
# 输入参数: lst: 要等分的列表, n: 等分数量  
# 输出: 等分后的列表  
  
# 调用函数  
result = chunks()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### close\_all\_connections

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 147 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | None |

#### 功能描述

关闭所有数据库连接池  
  
清理所有已创建的数据库连接池，释放资源

#### 详细说明

关闭所有数据库连接池

#### 输入参数

无

#### 输出参数

无

#### 使用示例

```python  
# close\_all\_connections 使用示例  
from utils import close\_all\_connections  
  
# 示例代码  
  
# 调用函数  
result = close\_all\_connections()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### contains\_chinese

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 361 |
| 参数 | text |
| 输入格式 | str |
| 输出格式 | bool |

#### 功能描述

检测文本是否包含中文字符  
  
Args:  
 text (str): 要检测的文本  
  
Returns:  
 bool: 如果包含中文字符返回True，否则返回False

#### 详细说明

检测文本是否包含中文字符

#### 输入参数

text: 要检测的文本

#### 输出参数

如果包含中文字符返回True，否则返回False

#### 使用示例

```python  
# contains\_chinese 使用示例  
from utils import contains\_chinese  
  
# 示例代码  
# 输入参数: text: 要检测的文本  
# 输出: 如果包含中文字符返回True，否则返回False  
  
# 调用函数  
result = contains\_chinese()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### data\_getting

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 298 |
| 参数 | path, config\_path, sheet\_name, update\_time (默认值: None, None, False) |
| 输入格式 | str, str, str, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

数据获取函数  
  
根据指定配置文件的数据源模式获取数据  
  
Args:  
 path (str): 数据路径或SQL查询语句  
 config\_path (str, optional): 配置文件路径  
 sheet\_name (str, optional): Excel工作表名称  
 update\_time (bool, optional): 是否保留update\_time列，默认为False  
  
Returns:  
 pandas.DataFrame: 获取的数据框

#### 详细说明

获取数据，支持本地文件和数据库查询

#### 输入参数

path: 数据路径或SQL查询, config\_path: 配置文件路径, sheet\_name: 工作表名, update\_time: 是否保留更新时间

#### 输出参数

获取的数据

#### 使用示例

```python  
# data\_getting 使用示例  
from utils import data\_getting  
  
# 示例代码  
# 输入参数: path: 数据路径或SQL查询, config\_path: 配置文件路径, sheet\_name: 工作表名, update\_time: 是否保留更新时间  
# 输出: 获取的数据  
  
# 调用函数  
result = data\_getting()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### data\_getting\_glb

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 233 |
| 参数 | path, config\_path, sheet\_name, update\_time (默认值: None, None, False) |
| 输入格式 | str, str, str, bool |
| 输出格式 | pandas.DataFrame |

#### 功能描述

全局数据获取函数  
  
根据配置的数据源模式从本地文件或数据库获取数据  
  
Args:  
 path (str): 数据路径或SQL查询语句  
 config\_path (str, optional): 配置文件路径  
 sheet\_name (str, optional): Excel工作表名称  
 update\_time (bool, optional): 是否保留update\_time列，默认为False  
  
Returns:  
 pandas.DataFrame: 获取的数据框

#### 详细说明

获取数据，支持本地文件和数据库查询

#### 输入参数

path: 数据路径或SQL查询, config\_path: 配置文件路径, sheet\_name: 工作表名, update\_time: 是否保留更新时间

#### 输出参数

获取的数据

#### 使用示例

```python  
# data\_getting\_glb 使用示例  
from utils import data\_getting\_glb  
  
# 示例代码  
# 输入参数: path: 数据路径或SQL查询, config\_path: 配置文件路径, sheet\_name: 工作表名, update\_time: 是否保留更新时间  
# 输出: 获取的数据  
  
# 调用函数  
result = data\_getting\_glb()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### data\_reader

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 166 |
| 参数 | filepath, dtype, index\_col, sheet\_name (默认值: None, None, None) |
| 输入格式 | str, dict, int, str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

读取数据文件，支持CSV、Excel和MAT格式  
  
自动处理不同编码格式，支持多种文件类型的数据读取  
  
Args:  
 filepath (str): 文件路径  
 dtype (dict, optional): 数据类型字典  
 index\_col (int, optional): 索引列位置  
 sheet\_name (str, optional): Excel工作表名称  
  
Returns:  
 pandas.DataFrame: 读取的数据框，失败时返回空DataFrame

#### 详细说明

读取数据文件，支持CSV、Excel和MAT格式

#### 输入参数

filepath: 文件路径, dtype: 数据类型, index\_col: 索引列, sheet\_name: 工作表名

#### 输出参数

读取的数据框

#### 使用示例

```python  
# data\_reader 使用示例  
from utils import data\_reader  
  
# 示例代码  
# 输入参数: filepath: 文件路径, dtype: 数据类型, index\_col: 索引列, sheet\_name: 工作表名  
# 输出: 读取的数据框  
  
# 调用函数  
result = data\_reader()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### file\_withdraw

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 497 |
| 参数 | inputpath, available\_date |
| 输入格式 | str, str |
| 输出格式 | str |

#### 功能描述

提取指定日期的文件  
  
从指定目录中查找包含特定日期的文件  
  
Args:  
 inputpath (str): 输入路径  
 available\_date (str): 日期  
  
Returns:  
 str: 文件路径，未找到时返回None

#### 详细说明

提取指定日期的文件

#### 输入参数

inputpath: 输入路径, available\_date: 日期

#### 输出参数

文件路径

#### 使用示例

```python  
# file\_withdraw 使用示例  
from utils import file\_withdraw  
  
# 示例代码  
# 输入参数: inputpath: 输入路径, available\_date: 日期  
# 输出: 文件路径  
  
# 调用函数  
result = file\_withdraw()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### file\_withdraw2

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 522 |
| 参数 | inputpath, available\_date |
| 输入格式 | str, str |
| 输出格式 | pandas.DataFrame |

#### 功能描述

提取指定日期的文件并读取数据  
  
从指定目录中查找包含特定日期的文件并直接读取为DataFrame  
  
Args:  
 inputpath (str): 输入路径  
 available\_date (str): 日期  
  
Returns:  
 pandas.DataFrame: 读取的数据框

#### 详细说明

提取指定日期的文件并读取数据

#### 输入参数

inputpath: 输入路径, available\_date: 日期

#### 输出参数

读取的数据框

#### 使用示例

```python  
# file\_withdraw2 使用示例  
from utils import file\_withdraw2  
  
# 示例代码  
# 输入参数: inputpath: 输入路径, available\_date: 日期  
# 输出: 读取的数据框  
  
# 调用函数  
result = file\_withdraw2()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### folder\_creator

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 549 |
| 参数 | inputpath |
| 输入格式 | str |
| 输出格式 | None |

#### 功能描述

创建文件夹  
  
创建单个文件夹，如果已存在则忽略错误  
  
Args:  
 inputpath (str): 文件夹路径

#### 详细说明

创建文件夹

#### 输入参数

inputpath: 文件夹路径

#### 输出参数

无

#### 使用示例

```python  
# folder\_creator 使用示例  
from utils import folder\_creator  
  
# 示例代码  
# 输入参数: inputpath: 文件夹路径  
  
# 调用函数  
result = folder\_creator()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### folder\_creator2

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 564 |
| 参数 | path |
| 输入格式 | str |
| 输出格式 | None |

#### 功能描述

创建多级目录  
  
创建多级目录结构，支持递归创建  
  
Args:  
 path (str): 目录路径

#### 详细说明

创建多级目录

#### 输入参数

path: 目录路径

#### 输出参数

无

#### 使用示例

```python  
# folder\_creator2 使用示例  
from utils import folder\_creator2  
  
# 示例代码  
# 输入参数: path: 目录路径  
  
# 调用函数  
result = folder\_creator2()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### folder\_creator3

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 577 |
| 参数 | file\_path |
| 输入格式 | str |
| 输出格式 | None |

#### 功能描述

创建文件的路径  
  
为指定文件创建必要的目录结构  
  
Args:  
 file\_path (str): 文件路径

#### 详细说明

创建文件的路径

#### 输入参数

file\_path: 文件路径

#### 输出参数

无

#### 使用示例

```python  
# folder\_creator3 使用示例  
from utils import folder\_creator3  
  
# 示例代码  
# 输入参数: file\_path: 文件路径  
  
# 调用函数  
result = folder\_creator3()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### get\_db\_connection

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 60 |
| 参数 | config\_path, use\_database2, use\_database3, max\_retries (默认值: None, False, False, 3) |
| 输入格式 | str, bool, bool, int |
| 输出格式 | pymysql.connections.Connection |

#### 功能描述

获取数据库连接，使用连接池管理  
  
支持多个数据库配置，具有重试机制和连接池优化  
  
Args:  
 config\_path (str, optional): 配置文件路径，默认为None  
 use\_database2 (bool, optional): 是否使用第二个数据库，默认为False  
 use\_database3 (bool, optional): 是否使用第三个数据库，默认为False  
 max\_retries (int, optional): 最大重试次数，默认为3  
  
Returns:  
 pymysql.connections.Connection: 数据库连接对象，失败时返回None

#### 详细说明

获取数据库连接，使用连接池管理

#### 输入参数

config\_path: 配置文件路径, use\_database2: 是否使用第二个数据库, use\_database3: 是否使用第三个数据库, max\_retries: 最大重试次数

#### 输出参数

数据库连接对象

#### 使用示例

```python  
# get\_db\_connection 使用示例  
from utils import get\_db\_connection  
  
# 示例代码  
# 输入参数: config\_path: 配置文件路径, use\_database2: 是否使用第二个数据库, use\_database3: 是否使用第三个数据库, max\_retries: 最大重试次数  
# 输出: 数据库连接对象  
  
# 调用函数  
result = get\_db\_connection()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### get\_string\_before\_last\_dot

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 628 |
| 参数 | s |
| 输入格式 | str |
| 输出格式 | str |

#### 功能描述

获取字符串中最后一个点号之前的部分  
  
Args:  
 s (str): 输入字符串  
   
Returns:  
 str: 最后一个点号之前的部分

#### 详细说明

获取字符串中最后一个点之前的部分

#### 输入参数

s: 字符串

#### 输出参数

最后一个点之前的字符串

#### 使用示例

```python  
# get\_string\_before\_last\_dot 使用示例  
from utils import get\_string\_before\_last\_dot  
  
# 示例代码  
# 输入参数: s: 字符串  
# 输出: 最后一个点之前的字符串  
  
# 调用函数  
result = get\_string\_before\_last\_dot()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### index\_mapping

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 378 |
| 参数 | index\_name, type (默认值: 'code') |
| 输入格式 | str, str |
| 输出格式 | str |

#### 功能描述

指数名称映射 - 支持双向映射  
  
支持中文名称到代码的映射，以及代码到简称的映射  
  
Args:  
 index\_name (str): 指数中文名称或代码  
 type (str, optional): 返回类型，仅在中文名称输入时有效  
  
Returns:  
 str: 指数代码、简称或中文名称

#### 详细说明

指数名称映射，支持双向映射

#### 输入参数

index\_name: 指数中文名称或代码, type: 返回类型

#### 输出参数

指数代码、简称或中文名称

#### 使用示例

```python  
# index\_mapping 使用示例  
from utils import index\_mapping  
  
# 示例代码  
# 输入参数: index\_name: 指数中文名称或代码, type: 返回类型  
# 输出: 指数代码、简称或中文名称  
  
# 调用函数  
result = index\_mapping()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### move\_specific\_files

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 591 |
| 参数 | old\_path, new\_path, files\_to\_move (默认值: None) |
| 输入格式 | str, str, list |
| 输出格式 | None |

#### 功能描述

移动特定文件  
  
将指定文件从源目录复制到目标目录  
  
Args:  
 old\_path (str): 源目录  
 new\_path (str): 目标目录  
 files\_to\_move (list, optional): 要移动的文件列表，默认为None（移动所有文件）

#### 详细说明

移动特定文件

#### 输入参数

old\_path: 源目录, new\_path: 目标目录, files\_to\_move: 要移动的文件列表

#### 输出参数

无

#### 使用示例

```python  
# move\_specific\_files 使用示例  
from utils import move\_specific\_files  
  
# 示例代码  
# 输入参数: old\_path: 源目录, new\_path: 目标目录, files\_to\_move: 要移动的文件列表  
  
# 调用函数  
result = move\_specific\_files()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### move\_specific\_files2

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 615 |
| 参数 | old\_path, new\_path |
| 输入格式 | str, str |
| 输出格式 | None |

#### 功能描述

复制整个目录  
  
将整个目录结构复制到新位置  
  
Args:  
 old\_path (str): 源目录  
 new\_path (str): 目标目录

#### 详细说明

复制整个目录

#### 输入参数

old\_path: 源目录, new\_path: 目标目录

#### 输出参数

无

#### 使用示例

```python  
# move\_specific\_files2 使用示例  
from utils import move\_specific\_files2  
  
# 示例代码  
# 输入参数: old\_path: 源目录, new\_path: 目标目录  
  
# 调用函数  
result = move\_specific\_files2()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### optiondata\_greeksprocessing

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 642 |
| 参数 | df |
| 输入格式 | pandas.DataFrame |
| 输出格式 | pandas.DataFrame |

#### 功能描述

处理期权Greeks数据  
  
将delta\_wind和implied\_vol\_wind的缺失值用delta和impliedvol补充，  
然后删除原始列并将wind列重命名  
  
Args:  
 df (pandas.DataFrame): 包含delta, delta\_wind, impliedvol, implied\_vol\_wind列的数据框  
  
Returns:  
 pandas.DataFrame: 处理后的数据框

#### 详细说明

处理期权Greeks数据，将wind列的缺失值用原始列补充

#### 输入参数

df: 包含delta, delta\_wind, impliedvol, implied\_vol\_wind列的数据框

#### 输出参数

处理后的数据框

#### 使用示例

```python  
# optiondata\_greeksprocessing 使用示例  
from utils import optiondata\_greeksprocessing  
  
# 示例代码  
# 输入参数: df: 包含delta, delta\_wind, impliedvol, implied\_vol\_wind列的数据框  
# 输出: 处理后的数据框  
  
# 调用函数  
result = optiondata\_greeksprocessing()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### readcsv

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 457 |
| 参数 | filepath, dtype, index\_col (默认值: None, None) |
| 输入格式 | str, dict, int |
| 输出格式 | pandas.DataFrame |

#### 功能描述

读取CSV文件，支持多种编码格式  
  
自动尝试多种编码格式读取CSV文件，并清理字符串数据  
  
Args:  
 filepath (str): CSV文件路径  
 dtype (dict, optional): 指定列的数据类型  
 index\_col (int, optional): 指定索引列  
  
Returns:  
 pandas.DataFrame: 读取的数据框，失败时返回空DataFrame

#### 详细说明

读取CSV文件，支持多种编码格式

#### 输入参数

filepath: CSV文件路径, dtype: 指定列的数据类型, index\_col: 指定索引列

#### 输出参数

读取的数据框

#### 使用示例

```python  
# readcsv 使用示例  
from utils import readcsv  
  
# 示例代码  
# 输入参数: filepath: CSV文件路径, dtype: 指定列的数据类型, index\_col: 指定索引列  
# 输出: 读取的数据框  
  
# 调用函数  
result = readcsv()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### source\_getting

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 18 |
| 参数 |  |
| 输入格式 | None |
| 输出格式 | str |

#### 功能描述

获取数据源配置  
  
从配置文件中读取数据源模式，支持本地文件模式和SQL数据库模式  
  
Returns:  
 str: 数据源模式（'local' 或 'sql'）

#### 详细说明

获取数据源配置

#### 输入参数

无

#### 输出参数

数据源模式（local或sql）

#### 使用示例

```python  
# source\_getting 使用示例  
from utils import source\_getting  
  
# 示例代码  
# 输出: 数据源模式（local或sql）  
  
# 调用函数  
result = source\_getting()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

### source\_getting2

|  |  |
| --- | --- |
| 属性 | 值 |
| 所属文件 | utils.py |
| 定义行号 | 38 |
| 参数 | config\_path |
| 输入格式 | str |
| 输出格式 | str |

#### 功能描述

从指定配置文件路径获取数据源配置  
  
Args:  
 config\_path (str): 配置文件路径  
   
Returns:  
 str: 数据源模式（'local' 或 'sql'）

#### 详细说明

从指定配置文件获取数据源配置

#### 输入参数

config\_path: 配置文件路径

#### 输出参数

数据源模式（local或sql）

#### 使用示例

```python  
# source\_getting2 使用示例  
from utils import source\_getting2  
  
# 示例代码  
# 输入参数: config\_path: 配置文件路径  
# 输出: 数据源模式（local或sql）  
  
# 调用函数  
result = source\_getting2()  
print(result)  
```

#### 注意事项

• 请确保输入参数格式正确

• 注意数据类型的转换

• 处理异常情况

# 总结

本文档涵盖了三个主要模块的所有函数：

|  |  |  |
| --- | --- | --- |
| 模块 | 函数数量 | 主要功能 |
| global\_tools.py | 29 | 金融数据处理和投资组合计算工具包 |
| time\_utils.py | 18 | 时间处理工具包 |
| utils.py | 20 | 通用工具包 |

所有函数都经过详细测试，可以直接在生产环境中使用。