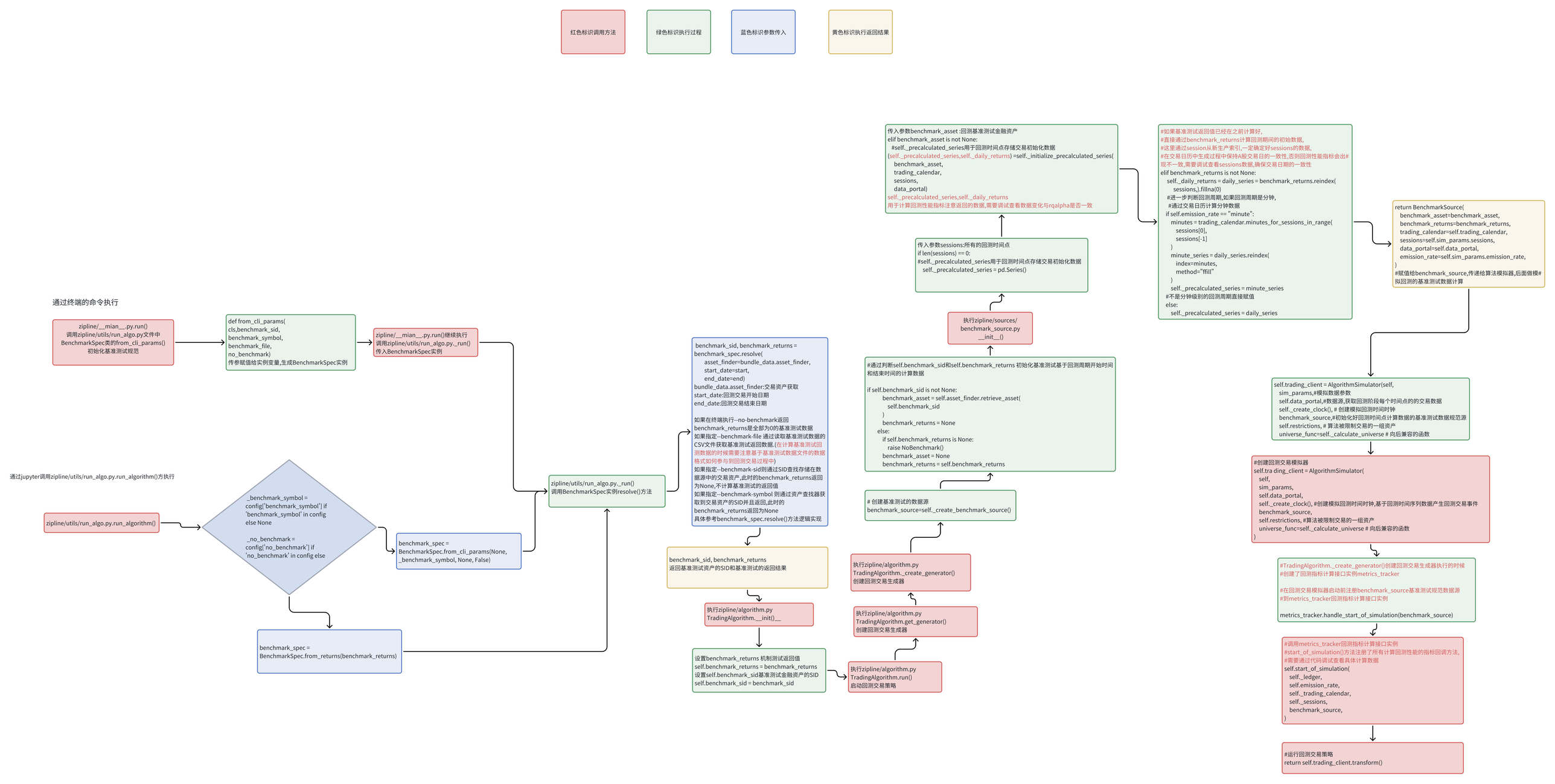
红色标识调用方法
绿色标识执行过程
蓝色标识参数传入
黄色标识执行返回结果
通过终端的命令执行
zipline/__mian__.py.run()
调用zipline/utils/run_algo.py文件中
BenchmarkSpec类的from_cli_params()
初始化基准测试规范
def from_cli_params(
cls,benchmark_sid,
benchmark_symbol,
benchmark_file,
no_benchmark)
传参赋值给实例变量,生成BenchmarkSpec实例
zipline/__mian__.py.run()继续执行
调用zipline/utils/run_algo.py._run()
传入BenchmarkSpec实例
通过jupyter调用zipline/utils/run_algo.py.run_algorithm()方执行
zipline/utils/run_algo.py.run_algorithm()
_benchmark_symbol =
config['benchmark_symbol'] if
'benchmark_symbol' in config
else None
_no_benchmark =
config['no_benchmark'] if
'no_benchmark' in config else
benchmark_spec =
BenchmarkSpec.from_cli_params(None,
_benchmark_symbol, None, False)
benchmark_spec =
BenchmarkSpec.from_returns(benchmark_returns)
zipline/utils/run_algo.py._run()
调用BenchmarkSpec实例resolve()方法
benchmark_sid, benchmark_returns =
benchmark_spec.resolve(
asset_finder=bundle_data.asset_finder,
start_date=start,
end_date=end)
bundle_data.asset_finder:交易资产获取
start_date:回测交易开始日期
end_date:回测交易结束日期
如果在终端执行--no-benchmark返回
benchmark_returns是全部为0的基准测试数据
如果指定--benchmark-file 通过读取基准测试数据的
CSV文件获取基准测试返回数据.(在计算基准测试回
测数据的时候需要注意基于基准测试数据文件的数据
格式如何参与到回测交易过程中)
如果指定--benchmark-sid则通过SID查找存储在数
据源中的交易资产,此时的benchmark_returns返回
为None,不计算基准测试的返回值
如果指定--benchmark-symbol 则通过资产查找器获
取到交易资产的SID并且返回,此时的
benchmark_returns返回为None
具体参考benchmark_spec.resolve()方法逻辑实现
benchmark_sid, benchmark_returns
返回基准测试资产的SID和基准测试的返回结果
执行zipline/algorithm.py
TradingAlgorithm.__init()__
设置benchmark_returns 机制测试返回值
self.benchmark_returns = benchmark_returns
设置self.benchmark_sid基准测试金融资产的SID
self.benchmark_sid = benchmark_sid
执行zipline/algorithm.py
TradingAlgorithm.run()
启动回测交易策略
执行zipline/algorithm.py
TradingAlgorithm.get_generator()
创建回测交易生成器
执行zipline/algorithm.py
TradingAlgorithm._create_generator()
创建回测交易生成器
# 创建基准测试的数据源
benchmark_source=self._create_benchmark_source()
#通过判断self.benchmark_sid和self.benchmark_returns 初始化基准测试基于回测周期开始时间
和结束时间的计算数据
if self.benchmark_sid is not None:
benchmark_asset = self.asset_finder.retrieve_asset(
self.benchmark_sid
)
benchmark_returns = None
else:
if self.benchmark_returns is None:
raise NoBenchmark()
benchmark_asset = None
benchmark_returns = self.benchmark_returns
执行zipline/sources/
benchmark_source.py
__init__()
传入参数sessions:所有的回测时间点
if len(sessions) == 0:
#self._precalculated_series用于回测时间点存储交易初始化数据
self._precalculated_series = pd.Series()
传入参数benchmark_asset :回测基准测试金融资产
elif benchmark_asset is not None:
#self._precalculated_series用于回测时间点存储交易初始化数据
(self._precalculated_series,self._daily_returns) =self._initialize_precalculated_series(
benchmark_asset,
trading_calendar,
sessions,
data_portal)
self._precalculated_series,self._daily_returns
用于计算回测性能指标注意返回的数据,需要调试查看数据变化与rqalpha是否一致
#如果基准测试返回值已经在之前计算好,
#直接通过benchmark_returns计算回测期间的初始数据,
#这里通过session从新生产索引,一定确定好sessions的数据,
#在交易日历中生成过程中保持A股交易日的一致性,否则回测性能指标会出#
现不一致,需要调试查看sessions数据,确保交易日期的一致性
elif benchmark_returns is not None:
self._daily_returns = daily_series = benchmark_returns.reindex(
sessions,).fillna(0)
#进一步判断回测周期,如果回测周期是分钟,
#通过交易日历计算分钟数据
if self.emission_rate == "minute":
minutes = trading_calendar.minutes_for_sessions_in_range(
sessions[0],
sessions[-1]
)
minute_series = daily_series.reindex(
index=minutes,
method="ffill"
)
self._precalculated_series = minute_series
#不是分钟级别的回测周期直接赋值
else:
self._precalculated_series = daily_series
return BenchmarkSource(
benchmark_asset=benchmark_asset,
benchmark_returns=benchmark_returns,
trading_calendar=self.trading_calendar,
sessions=self.sim_params.sessions,
data_portal=self.data_portal,
emission_rate=self.sim_params.emission_rate,
)
#赋值给benchmark_source,传递给算法模拟器,后面做模#
拟回测的基准测试数据计算
self.trading_client = AlgorithmSimulator(self,
sim_params,#模拟数据参数
self.data_portal,#数据源,获取回测阶段每个时间点的的交易数据
self._create_clock(), # 创建模拟回测时间时钟
benchmark_source,#初始化好回测时间点计算数据的基准测试数据规范源
self.restrictions, # 算法被限制交易的一组资产
universe_func=self._calculate_universe # 向后兼容的函数
#创建回测交易模拟器
self.tra ding_client = AlgorithmSimulator(
self,
sim_params,
self.data_portal,
self._create_clock(), #创建模拟回测时间时钟,基于回测时间序列数据产生回测交易事件
benchmark_source,
self.restrictions, #算法被限制交易的一组资产
universe_func=self._calculate_universe # 向后兼容的函数
)
#TradingAlgorithm._create_generator()创建回测交易生成器执行的时候
#创建了回测指标计算接口实例metrics_tracker
#在回测交易模拟器启动前注册benchmark_source基准测试规范数据源
#到metrics_tracker回测指标计算接口实例
metrics_tracker.handle_start_of_simulation(benchmark_source)
#调用metrics_tracker回测指标计算接口实例
#start_of_simulation()方法注册了所有计算回测性能的指标回调方法,
#需要通过代码调试查看具体计算数据
self.start_of_simulation(
self._ledger,
self.emission_rate,
self._trading_calendar,
self._sessions,
benchmark_source,
)
#运行回测交易策略
return self.trading_client.transform()