本文档截取自课程作业文档的需求分析与用例分析部分。由于时间关系没有做相应的调整，因此文中可能出现一些没头没尾的内容，请见谅。

1 回测用例

1. 简要说明

用户可以通过 GUI 按照默认参数执行回测，也可以通过脚本方式执行回测，但无论使用哪种方式，若用户想要自己设计股票策略，都必须编写 Python 代码，否则只能选择系统提供的策略（主要用于演示系统功能，实际意义有限）。

针对用户自行编写的策略，在此需要做一些特别说明。自定义策略需要继承一个名为 BaseStrategy 的抽象基类，利用该抽象基类，用户可以实现两个功能。

其一，该抽象基类以类属性的方式保存了用于获取数据的接口，用户可以利用该接口获取沪深 A 股行情数据、上市公司财务数据、场内基金数据、指数数据、期货数据、期权数据、场外基金信息、宏观经济数据、债券数据、舆情数据，总共 10 大类别 300 余项数据。该数据接口大部分集成自 JQData，在 www.joinquant.com 网站上附有详细的文档，用户应该参阅该文档来获取自己想要的信息。需要注意的是，由于本项目将 JQData 的接口与其他数据的接口统一进行了封装，JQData 的函数只有加上“self.api.”的目的对象前缀才能够被正常调用，除此之外如函数签名以及异常处理等与 JQData 没有区别。

其二，BaseStrategy 规定了一个名为 handleBar 的抽象方法，因此所有的子类都必须实现 handleBar 方法，用户自行编写的策略类也不例外。每一个回测周期该方法都会被回测类运行一次，因此 handleBar 是用户实现策略的主要地方。

当用户需要模拟股票的买入、卖出或平仓时，应该实例化一个 SignalEvent 事件，并为其传递恰当的初始化参数。SignalEvent 事件的定义保存在 backtest 目录下的 event.py 中，需要用户自行导入。SignalEvent 被实例化后需要加入 events_queue 队列才能实现股票的买入或卖出模拟。

对于图形界面运行方式，用户可以在窗口中选择内置策略或在提示面板输入自定义策略类所在的 py 文件路径，然后点按回测按钮，即可按照默认参数回测（模拟交易）。对于脚本运行方式，用户可以调整更多的选项。用户首先需要实例化一个名为 Backtest 的回测类并指定相应的参数，策略类型为不可缺省的参数，而回测频率、开始时间、结束时间、手续费、滑点、起始资金和业绩基准等参数可以自由设置。Backtest 类的定义保存在 backtest 目录下的 backtest.py 文件中，同样地，该类需要用户自行导入到程序入口当中。

本系统目前可以模拟日级别或更长调仓周期的多策略和多标的股票交易。用户既可以选择系统内置的策略，也可以自行编写策略代码。基于事件驱动引擎，本系统模拟了从股票市场数

据更新开始，到交易信号的计算和发出，再到资产头寸管理，最后在交易所完成下单的周而复始的循环。系统最终生成可视化的回测报告，从多个维度评价策略的表现。

2. 事件流

① 基本事件流

用例开始于用户已经选择了内置策略，或填写了保存有自定义策略的 py 文件的路径，然后点击“回测”按钮。

A. 确定回测策略：

A1. 未选择内置策略，也未填写文件路径；

A2. 未选择内置策略，且文件路径不存在；

A3. 未选择内置策略，且 py 文件中不包含 BaseStrategy 的子类；

A4. 未选择内置策略，且 BaseStrategy 的子类中没有实现 handleBar 方法；

B. 系统模拟股票市场数据更新；

C. 系统计算并发出交易信号；

D. 系统按照风险控制规则管理资产头寸；

E. 系统模拟交易所下单的过程；

F. 系统不断重复上述 ABCD 四项步骤，直到遍历完回测区间；

G. 系统生成可视化回测报告。

② 后备事件流

A1. 未选择内置策略，也未填写文件路径：

- 提示“未选择策略!”；
- 结束用例，不执行后续的任何事件流；

A2. 未选择内置策略，且文件路径不存在；

- 提示“文件路径不存在!”；
- 结束用例，不执行后续的任何事件流；

A3. 未选择内置策略，且 py 文件中不包含 BaseStrategy 的子类；

- 提示“未实现策略类!”；
- 结束用例，不执行后续的任何事件流；

A4. 未选择内置策略，且 BaseStrategy 的子类中没有实现 handleBar 方法；

- 提示“策略类未实现 handleBar 方法！”；
- 结束用例，不执行后续的任何事件流；

3. 特殊要求

无。

4. 前置条件

用例开始于用户已经填写了保存有自定义策略的 py 文件的路径，然后点击“回测”按钮。

5. 后置条件

无。

6. 活动图

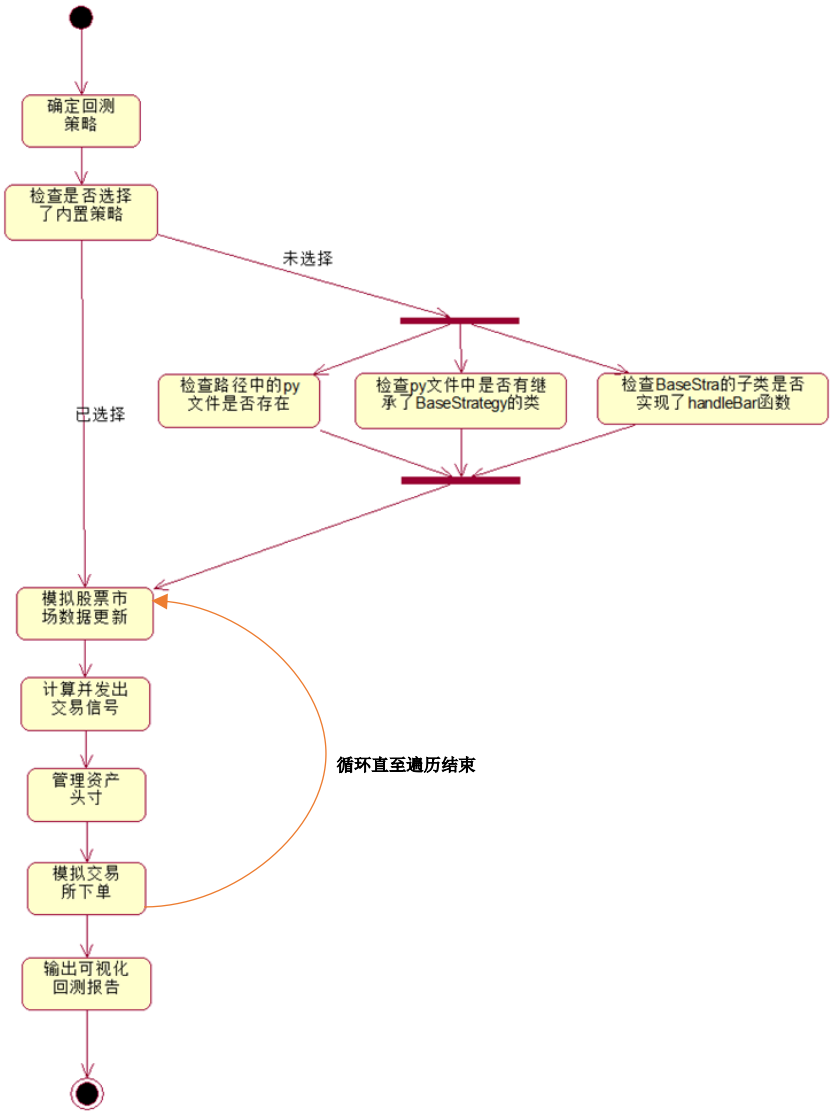

图 1 回测用例活动图

2 执行事件驱动回测用例类的析取

根据“执行事件驱动回测”的用例规约，经分析可得该用例中的三种类。用户可以通过 GUI 按照默认参数执行回测，也可以通过脚本方式执行回测，但无论使用哪种方式，若用户想要自己设计股票策略，都必须编写代码，否则只能选择系统提供的演示策略。

边界类：Frame。Frame 中包含了回测策略选择面板。用户在面板中选择需要进行的回测策略。

控制类：

1. Backtest 类：包含 Strategy、DataHandler、Portfolio、ExecutionHandler 和 Performance 对象或者其子类的对象的一个组合类，用于在宏观上控制回测的执行，具体的操作则隐藏在各个组件的实例方法中，回测参数的调节也在此类当中控制；
2. BaseStrategy 类：所有策略类的抽象基类，当前的作用仅限于规定子类必须包含一个名为 handleBar 的方法，以及将数据接口保存为类属性以便子类调用，但从提高系统的可扩展性出发仍然保留了此类；
3. BuySingleStockEverydayStrategy 类：BaseStrategy 的子类，回测期间每日买入固定数量的制定股票，主要用于测试系统是否能够正常运行，并无实际意义；
4. BuyMultipleStockEverydayStrategy 类：BaseStrategy 的子类，回测期间每日买入固定数量的任意只股票，主要用于测试系统是否能够正常运行，并无实际意义；
5. DoubleMovingAverageStrategy 类：BaseStrategy 的子类，当短均线上穿场均先时买入股票，当短均线下穿长均线时卖出所有股票。该策略有中文名为双均线策略，用于代表以价格和交易量等行情信息在单只股票上操作的一类策略。该策略主要用于测试系统功能，以及是否能够正常调用行情数据，实际意义并不大；
6. PEStrategy 类：BaseStrategy 的子类，每月在通过市值和应收筛选的股票池内买入动态市盈率最低的若干只股票。PE（动态市盈率）是因子选股中的一个基本因子，通常认为 PE 较低的股票更有性价比。该策略主要用于测试系统功能，是否能够正常调用财务数据，以及月度级别的调参频率，有一些实际意义；
7. MyStrategy 类（名称任意）：BaseStrategy 的子类，这是一个由用户自定义的策略类，用户在其中编写策略代码。MyStrategy 类至少应该包含一个 handleBar 方法，用户可以在其中决定回测期间每一天的行为，具体要实现什么样的股票策略取决于用户的想法。支持任意类型的策略是本项目的关键功能之一；
8. DataHandler 类：所有 data handler 类的抽象基类，当前的作用仅限于规定子类必须包

含一个名为 updateBars 的方法，以及将数据接口保存为类属性以便子类调用，但从提高系统的可扩展性出发仍然保留了此类。针对不同的数据源需要有不同的数据预处理操作，DataHandler 规定了这些 data handler 最重要的特征，即每日收盘后向事件队列当中添加一个 MarketEvent 事件，触发了后续的一系列操作；

9. RQBundleDataHandler 类：DataHandle 的子类，针对米筐数据源的一个 datahandler，负责从 Database 类当中获取数据，根据当前回测参数有针对性地完成数据预处理操作，以及作为回测部分其他组件的数据接口，从而将其他组件与底层数据库 Database 隔离；
10. ExecutionHandler 类：所有 execution handler 的抽象基类，当前的作用仅限于规定子类必须包含一个名为 executeOrder 的方法，但从提高系统的可扩展性出发仍然保留了此类；
11. SimulatedExecutionHandler 类：ExecutionHandler 的子类，用于模拟在交易所下单的过程，判断股票是否可以交易，模拟只能部分买入的情况，计算滑点价格和各种交易成本，以向事件队列发送一个 FillEvent 结束；
12. Performance 类：根据回测期间的投资组合，计算一系列绩效指标，包括总收益、年化收益、策略 α 、策略 β 、夏普比率、最大回撤、波动率、信息比率和回测曲线，并且与基准收益作比较；
13. DataAPI 类：Env 的子类，是用户通过脚本查询各类数据的接口。该类对 Database 类的所有函数都进行了一次封装，从而将底层数据库与用户隔离；

实体类：

1. Event 类：所有事件类的抽象基类，当前的作用仅限于规定所有子类都必须实现__repr__魔法方法和__eq__魔法方法，用于在回测过程中输出事件和事件优先级的比较，但从提高系统的可扩展性出发仍然保留了此类；
2. MarketEvent 类：Event 类的子类，包含事件类型、优先级、当天的时间戳和市场行情，由 data handler 发出，被回测类取出后分配给 Portfolio 更新投资组合价值，以及分配给策略类决定是否要发出交易信号。MarketEvent 的优先级处于较低一级；
3. SignalEvent 类：Event 类的子类，包含事件类型、优先级、信号发出的时间戳、交易标的名称、交易方向、交易数量和下单时间，由策略类发出，被回测类取出后分配给 Portfolio 做仓位管理。SignalEvent 的优先级处于较低一级；
4. OrderEvent 类：Event 类的子类，包含事件类型、优先级、订单时间戳、交易标的、交易方向、交易数量和下单时间，由 Portfolio 类发出，被回测类取出后分配给 execution 模拟交易所的下单过程。OrderEvent 表示经过仓位管理后用户拟进行交易的订单信息，

只有继续经过 execution 的处理，才能够转化为真实的交易订单。OrderEvent 的优先级处于较高的一级；

5. FillEvent 类：Event 类的子类，包含事件类型、优先级、订单时间戳、交易标的、交易方向、交易成本、交易数量、各类过程成本合计和订单时间，由 execution 发出，被回测类取出后分配给 Portfolio 更新证券账户。FillEvent 的优先级处于较高的一级；
6. PriorityQueue 类：标准库当中的优先级队列，优先弹出级别较高的元素，只有对于同级别的元素，才按先进先出的规则弹出。PriorityQueue 是事件驱动引擎的容器，对于回测而言，采用向量化的设计也是可行的，而且由于事件驱动设计引入了新的复杂度，所以事件驱动并非必要，但对于实盘交易而言，向量化交易不可行，必须采用事件驱动设计。为了提升系统的可扩展性，同时也为了与业界标准接轨，回测系统最终采用了事件驱动设计。与简单的事件驱动引擎不同，本系统还规定了各类事件的优先级，将 OrderEvent 和 FillEvent 的优先级置于 MarketEvent 和 SignalEven 之前，目的是应对策略类在一天当中同时发出多个信号的情况，这样一来，只有当上一个 SignalEvent 被 Portfolio 和 execution 处理完，此时证券账户被成功更新，才展开下一个 SignalEvent 的处理，避免了不同信号之间可能存在的冲突。
7. Portfolio 类：用于记录回测期间的投资组合，同时也充当模拟的证券账户进行仓位管理，仓位管理的规则过于复杂，不在此详述；
8. Database 类：本系统的底层数据库，负责从米筐网站上下载数据包、解压数据包、读取数据、股票池初选、复权和其他数据预处理操作，同时集成了聚宽 JQData 的所有数据接口。严格上来说，Database 类应该拆分为更小的实体类，直到每一个实体类只表示一种数据结构，本项目将这些小的实体类合并为一个大类，有如下原因。其一，历史以来的金融数据量过大，因此不能做本地存储，只能动态地从网络上获取，用户需要时再下载，所以小实体类如果存在，很可能只包含唯一一个用于获取数据的函数；其二，除了来自米筐的数据以外，其他数据的调用直接使用了 jqdatasdk 库的结果，如果小实体类存在，其唯一的方法也仅仅是调用第三方库的函数，返回第三方库的返回值罢了；其三，数据类型可以达到上百种之多，为每一种数据设计一个实体类表示没有可操作性。基于以上原因，本项目将所有的数据项囊括在一个 Database 类当中，尽管此设计与 MVC 标准存在冲突；
9. Env 类：存放环境信息，当前作用仅限于实例化一个 Database 类的实例，并完成所有加载操作，有此设计是因为一方面 Database 对象内存消耗严重，加载耗时非常长，所以系统运行过程中只能创建一个 Database 的实例，另一方面数据库处于系统底层，不应该以全局变量的形式暴露，所以最终通过继承的方式解决。

以上 9 中类的组合即为关键抽象中的 BackTestModel。

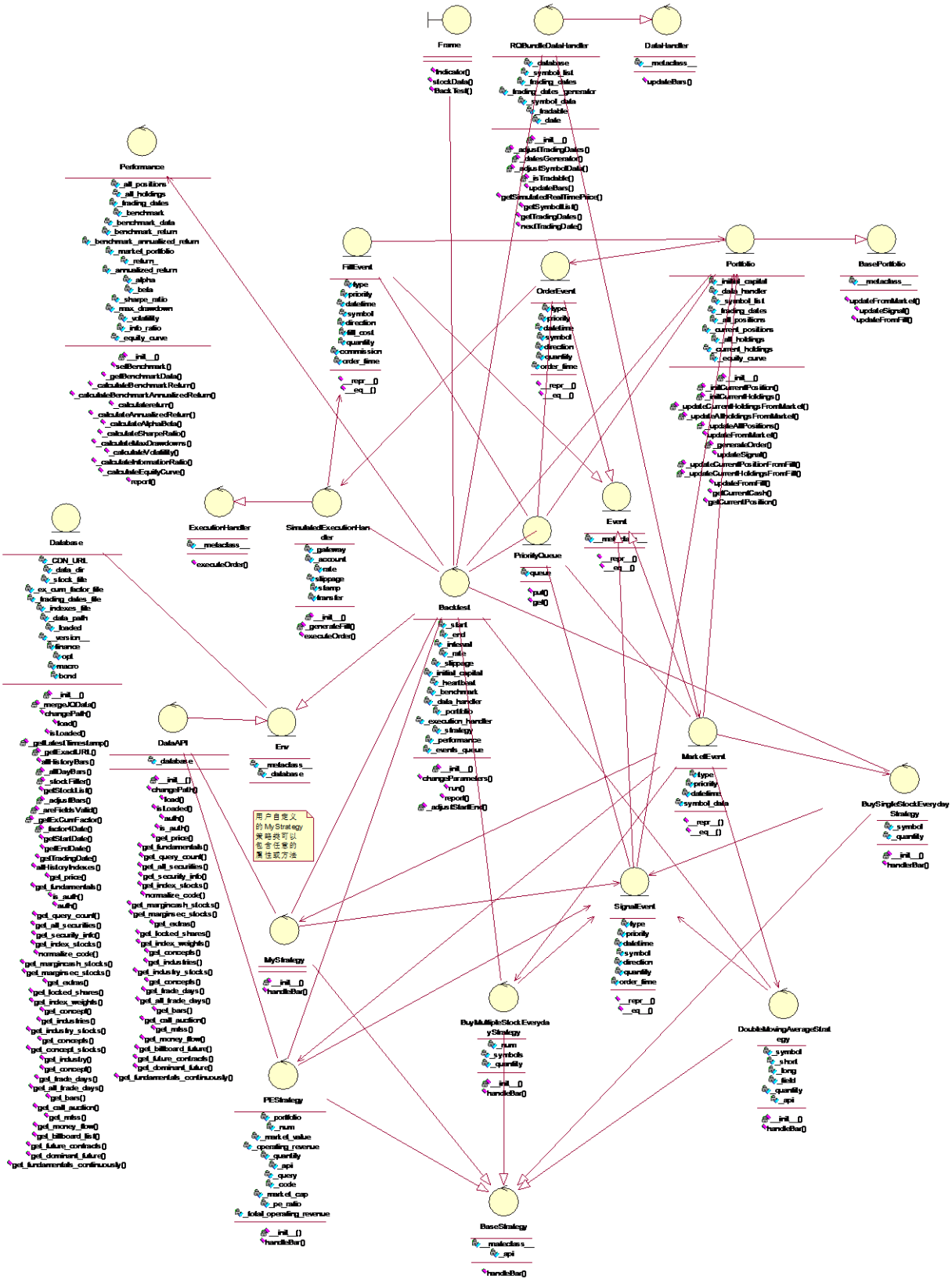

图 3-5 执行事件驱动回测用例析取图

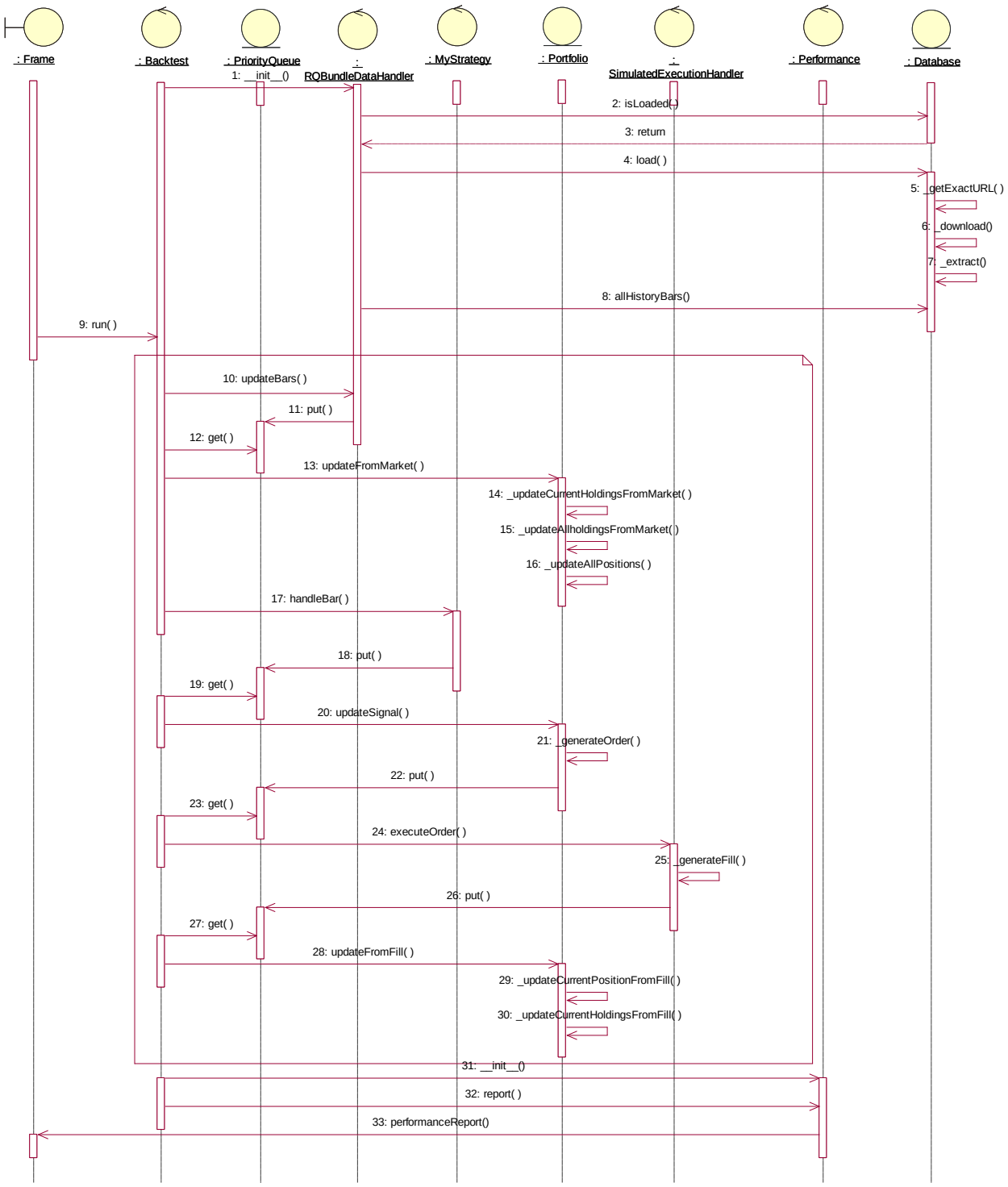

图 3-6 执行事件驱动回测用例时序图

由回测用例的类图不难想象，回测的执行过程较为复杂，同时也是因为 IBM Rational Rose 的时序图无法绘制条件结构和循环结构，笔者对实际的程序执行过程进行了大幅度的简化，去掉了繁琐的数据预处理、异常处理和一众初始化操作，只保留了回测的核心操作，同时把部分选择结构变成了顺序执行结构，除此之外，还把部分循环结构省略，用注释框表示剩余

的循环结构，从而得到了如上所示的时序图。

当用户点击界面的回测按钮时，本系统首先对所涉及的所有控制类和实体类进行初始化，其中以 RQBundleDataHandler 和 Database 类的操作为主。Database 类的初始化包括获取下载链接、下载和解压缩 2005 年以来的全部 A 股行情数据，下载行情数据和解压缩的时间因网速和处理器而有所不同。RQBundleDataHandler 的初始化将会从 Database 中获取行情数据、交易日列表、停牌信息和全市场组合，并对行情数据进行前向复权。

界面类随后调用 Backtest 类的 run 函数，该函数控制了回测的执行过程，其核心是两个 while 循环和一个优先级队列，它们在时序图中被一个注释框所包裹。run 函数按照事件驱动的逻辑进行设计，不断地从事件队列中获取事件，根据事件所属的类型调用不同控制类的相关函数进行处理，新的事件有可能在处理过程中被添加到事件队列中，从而形成了内层循环。内层循环的结束代表已完成一个交易日的所有操作，这将触发 RQBundleDataHandler 的 updateBar 函数往事件队列当中添加新的 MarketEvent 市场事件，该事件代表了新的一个交易日的产生，并紧接一个内层循环。上述过程循环往复，直到回测区间内的所有交易日都遍历完毕。

回测结束将会返回一个 Performance 对象，该对象的 report 方法根据回测区间内投资组合的价值变化计算出总收益、年化收益、策略 α 、策略 β 、夏普比率、最大回撤、波动率、信息比率和净值曲线，并和基准收益进行比对，最后调用界面类的 performanceReport 函数将结果可视化输出。