分布式数据库系统设计
项目技术报告

组长：周孟莹（19110240001）

组员：方睿钰 （19210240002）、章苏尧 （19212010032）

本报告撰写人：方睿钰

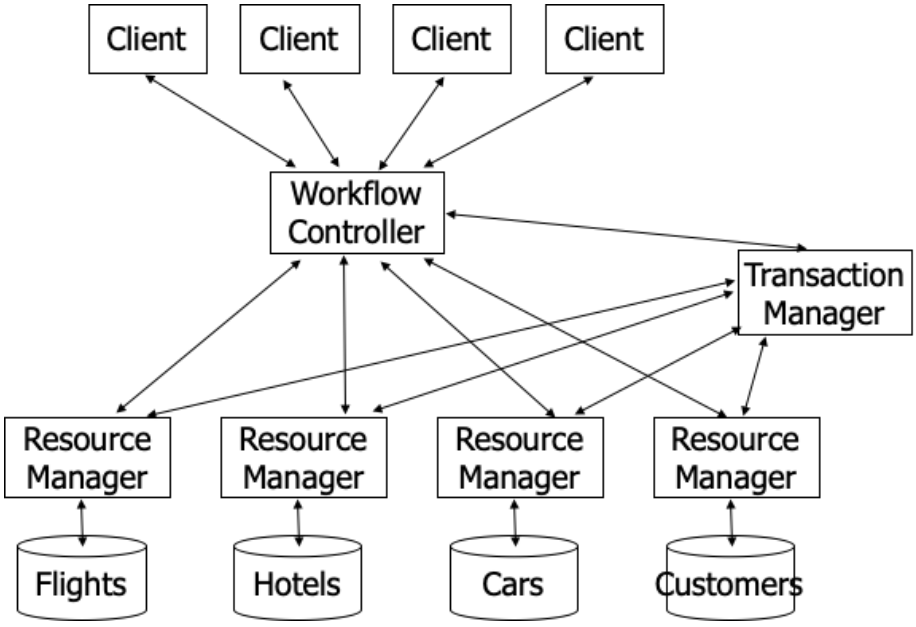
分工：

成员	分工	工作量
周孟莹 19110240001	负责协调整体项目、实现细化 Client 类、设计测试用例、撰写测试用例说明	33%
方睿钰 19210240002	实现 WorkflowController 类、编写 ReadMe 文件、撰写测试用例说明	33%
章苏尧 19212010032	实现 ResourceManager 类、调试测试用例、撰写 server 启动脚本和测试脚本	33%

一、 项目设计与整体架构

本项目实现的是一个简单的分布式旅游预订系统，其中主要的数据主体为 Customer、Flight、Car、Room。Customer 可在系统上对其他三个主体进行预订和取消预订操作，另每个主体都设有增删改查的操作。

作为分布式数据系统，从设想上来看系统的各个组件以及底层数据存储都可在不同的机器上存在，因此本项目组件间的接口调用均采用 JavaRMI 模拟远程调用。本项目组件主要分为 WorkflowController（WC），TransactionManager（TM），ResourceManager（RM）三大部分，其中 WC 负责最外层的用户调用，通过与 TM 和 RM 的交互实现业务逻辑，使得 TM 和 RM 对用户透明；TM 负责管理事务，保证事务的 ACID 特性，负责事务的流程、处理错误和异常、事务回滚等；RM 则是最底层接触数据的部分，每个数据主体都由对应的资源管理器进行访问操作，实现增删改查，确保对数据持久性。



二、 文件目录结构

- src
  - lockmgr: 已提供的实现好的锁管理器
  - test: Java 编写实现的测试用例
    - data: 运行生成的数据文件夹
    - TestFileObject.java: [TODO]
    - TestManager.java: 测试主程序
    - Makefile
    - 其它 Java 程序: 测试用例
  - transaction: 主要实现部分
    - data: 运行生成的数据文件夹
    - exceptions: 自定义的可能异常
    - models: 数据实体定义，包括接口 ResourceItem 与类 Car, Customer, Flight, Hotel, Reservation, ReservationKey
    - Client.java: 脚本测试时调用的客户端
    - MyClient.java: 默认简单客户端
    - ResourceManager.java: RM 接口
    - ResourceManagerImpl.java: RM 接口实现
    - RMTable.java: 数据表
    - TransactionManager.java: TM 接口（主要修改文件）
    - TransactionManagerImpl.java: TM 接口实现（主要修改文件）
    - Utils.java: 工具类，用于简化代码
    - WorkflowController.java: WC 接口
    - WorkflowControllerImpl.java: WC 接口实现（主要修改文件）
    - Makefile
- doc
  - 测试用例说明
  - 项目分工说明
  - 项目技术报告-周孟莹
  - 项目技术报告-方睿钰
  - 项目技术报告-章苏尧
- README.md
- run_server.sh
- run_test.sh
- stop_server.sh

三、 WC 和 TM 的具体实现

3.1 WC 的实现

3.1.1 WC 的工作流程

WC(WorkflowController)是整个系统中负责对外接口的部分，用户通过调用一系列 WC 的接口进行对数据系统的调用。WC 为用户提供针对数据主题的增加、删除、改动、查询、预定等操作，并向用户返回布尔值提示操作是否成功。

WC 在内部实现中通过对业务逻辑进行拆分，调用不同的接口实现指定业务。针对事务的 start/commit/abort, WC 会根据事务的唯一编号 xid 调用对应的 TM 接口，而针对数据的操作，则会调用对用的 RM 接口实现，其中针对每次数据的操作也需要提供事务的唯一编号 xid，以便后续 TM 对事务的管理。

3.1.2 WC 的异常检测与处理

针对本身宕机的情况，在 WC 中，每个事务都保留一个事务编号 xid（由 TM 生成和管理），并在文件中保留 xid 的集合，确保从故障中恢复后可以恢复所有未被完成的事务。

针对操作执行过程中可能抛出的异常，需要处理一般是 DeadLockException，在 WC 中进行捕捉后会直接调用 TM 终止事务，TM 会负责事务的回滚，之后 WC 会抛出 TransactionAbortionException 提示用户事务终止。

针对其他异常情况，例如其他组件链接不上，WC 会抛出 RemoteException。 若用户采用不存在 xid 提交操作，WC 会抛出

InvalidTransactionException。

另外，针对测试中需要出现的模拟宕机操作，WC 也提供了一系列接口，接口提供了 RM、TM、WC 在不同时刻的宕机模拟，为后续的测试提供方便。

组件启动接口		
Start	启动一个事务 id	start
Commit	提交一个事务	commit
Abort	终止一个事务	abort
基本业务逻辑接口		
Add*	添加 XX	addFlight, addRooms, addCars, newCustomer
Query*	查询 XX	queryFlight, queryFlightPrice, queryRooms, queryRoomsPrice, queryCars, queryCarsPrice, queryCustomerBill
Delete*	删除 XX	deleteFlight, addRooms, deleteRooms, addCars, deleteCars, deleteCustomer
Reserve*	预订 XX	reserveFlight, reserveCar, reserveRoom
异常模拟接口		
Die*	XX 崩溃	dieNow, dieRMAfterEnlist, dieRMBeforePrepare, dieRMAfterPrepare, dieTMBeforeCommit, dieTMAfterCommit, dieRMBeforeCommit, dieRMBeforeAbort

3.2 TM 的实现

3.2.1 TM 的工作流程

TM 是一个集中式两阶段提交的协调器，它需要管理所有的事务。TM 的工作是协调事务提交，RM 则是事务的执行者，事务能否 commit 成功需要 TM 做出判断，而最终的判断是基于 RM 给出的答复的基础上的。它们之间的交互分为以下阶段：

在一个事务被创建时，相关的 RM 也会被启动，这时 RM 会调用 TM 的 enlist 声明自己所参与的事务。

在事务提交时，第一阶段的提交，TM 会逐个询问参与该事务的 RM 是否准备好提交事务，即调用 RM 的接口 prepare 来得到答复，若 RM 本身没有检测到死锁等异常，则 RM 会回答 True，若所有 RM 都回到 True，则 TM 会判断该事务可以提交，如果不是，则该事务终止，并将该事务从所有参与的 RM 上负责的事务列表中移除。第二个阶段的提交，TM 这时已经确保所有的 RM 都准备好了提交该事务，则 TM 会调用 RM 的 commit 方法，使 RM 将改动保存至磁盘中，是该事务所做的改动持久化。

3.2.2 TM 的异常检测与处理

在第一阶段的提交过程中，RM 如果不回答 True，则事务终止，并且从所有 RM 的列表中移除，这样确保该事务所做的任何改动都无法实现。在第二阶段中若 RM 宕机，则需要将该 RM 从对应的事务列表中移除，等重启成功后再将该事务 id 加入 RM 列表中重新执行。TM 同时也实现了文件的 log 保存事务的状态，这样可在 TM 本身宕机后快速恢复事务的状态和 RM 对应的列表。

四、 测试用例编写

我们编写了 4 种类型的测试用例分别检测系统的事务处理、增删改查基础功能、宕机处理、锁处理。详细的内容可见 测试用例说明 文档。