项目技术报告

1 成员及分工

20210240009 许元武：负责 TransactionManager 和部分测试用例的编写

20210240007 李炳嘉：负责 WorkflowController 和部分测试用例的编写

2 功能实现

2.1 实体类的实现

本项目将原项目中的 Reservation，ReservationKey，ResourceItem 三个文件移到了 src/transaction/entity 包下便于统一管理，并在此基础上实现了 Car，Customer，Flight，Hotel 四个实体类，与 Reservation 一起分别对应系统存储的五张表。四个类均实现了 ResourceItem 接口，WorkflowControllerImpl 通过读取与存储这些实体类最终实现对不同实体的增删改查以及预订操作，同时通过统一的 ResourceItem 接口简化了对订单的操作。

以 Flight 类为例，类内包含 flightNum，price，numSeats，numAvail，isDeleted 五个属性，分别对应航班号，价格，座位数，剩余座位数及对象是否被删除，同时实现了对不同属性的基础操作。如用于增加航班订单的 addResv()函数，该函数首先检查剩余座位数是否大于 0，若满足条件则将座位数减一，否则订单失败。其余实体类的实现均与 Flight 类类似。

2.2 数据操作实现

WorkFlowController 接口中定义了对实体类的所有数据操作函数，流程控制器 WorkFlowControllerImpl 通过实现该接口，具体实现了对数据的操作功能。

首先，为了简化代码，我们定义了 queryItem()函数。由于 WorkFlowControllerImpl 针对不同的数据表使用不同的 ResourceManager（如 rmFlights）进行操作，queryItem()通过指定 ResourceManager 及主键内容，调用所指定 ResourceManager 中的 query 函数在对应数据表中查询相关内容，并以 ResourceItem 的形式返回。其中异常处理部分将在后文介绍。

```
public ResourceItem queryItem(ResourceManager rm, int xid, String key)
        throws RemoteException,
        TransactionAbortedException,
        InvalidTransactionException {
    if (!xids.contains(xid))
        throw new InvalidTransactionException(xid, "");
    ResourceItem resourceItem;
    try {
        resourceItem = rm.query(xid, rm.getID(), key);
    } catch (DeadlockException e) {
        abort(xid);
        throw new TransactionAbortedException(xid, e.getMessage());
    }
    return resourceItem;
}
```

该类中具体的数据操作最终均通过 queryItem()函数返回的 ResourceItem 对象实现。此处以 addCars()为例。

```
public boolean addCars(int xid, String location, int numCars, int price)
        throws RemoteException,
        TransactionAbortedException,
        InvalidTransactionException {
    if (!xids.contains(xid))
        throw new InvalidTransactionException(xid, "addCars");
    if (location == null)
        return false;
    if (numCars < 0)
        return false;
    ResourceItem resourceItem = queryItem(rmCars, xid, location);
    if (resourceItem == null) {
        price = Math.max(0, price);
        Car car = new Car(location, price, numCars);
        try {
            return rmCars.insert(xid, rmCars.getID(), car);
        } catch (DeadlockException e) {
            abort(xid);
            throw new TransactionAbortedException(xid, e.getMessage());
        }
    } else {
        Car car = (Car) resourceItem;
        car.addCars(numCars);
        if (price >= 0)
            car.setPrice(price);
        try {
            return rmCars.update(xid, rmCars.getID(), location, car);
        } catch (DeadlockException e) {
            abort(xid);
            throw new TransactionAbortedException(xid, e.getMessage());
        }
    }
}
```

该函数首先通过 queryItem()函数，指定 rmCars 及对应主键 location，在 Cars 数据表中查找是否包含该条数据。若返回的 ResourceItem 对象为空，说明数据表中没有该数据，此时我们新建一个 Car 对象并设置对应信息，通过 rmCars.insert()方法在数据表中插入对应内容。若返回的 ResourceItem 对象非空，说明数据表中已有相关数据，此时 ResouceItem 中已包含相关信息，我们只需在该对象上进行更新，并通过 RmCars.update()方法在数据表中更新对应的信息段即可。该方法中的异常处理部分将在后文进行介绍。其余数据操作方法均与此方法类似，不再赘述。

2.3 流程实现

数据操作函数具体实现了对不同数据表的增删改查等不同命令，而一个事务往往是由许多具体的命令构成的。因此，在 WorkFlowController 中还定义了如 start()，commit()，abort()等函数，对单个事务进行流程上的操作。

我们在 WorkFlowControllerImpl 中定义了一个哈希表 xids 用于储存事务号，start()方法代表新建一个事务，该方法通过调用 TransactionManager 中的 start()方法返回一个新的事务号并将其添加到 xids 中。此后该事务的所有数据操作均需绑定该事务号进行。abort()方法用于终止一个事务，通过调用 TransactionManager 中的 abort()方法实现。并将该事

务号从 xids 中删除。Commit()函数首先检查 xids 中是否存在所要提交的事务号，并通过调用 TransactionManager 中的 commit()方法进行提交，提交成功后将该事务号从 xids 中删除。

此外，WorkFlowController 中还定义了一系列 die 函数用于模拟 ResourceManager 及 TransactionManager 在不同时间段的宕机情况。为了简化代码，我们定义了一个 dieRMByTime()方法，通过指定 ResourceManager 及具体宕机时段，调用对应 ResourceManager 的 setDieTime()方法具体实现。针对 TransactionManager,通过调用 TransactionManager 的 setDieTime()方法进行实现。

3 异常处理

3.1 Wc 宕机

针对可能存在的 WorkFlowController 宕机问题，本项目通过存储哈希表 xids 保存当前事务情况。我们在 Util 类中定义了 storeObject()以及 loadObject()方法用于在磁盘上保存和读取 xids 对象，并在 WorkFlowControllerImpl 中定义了 recover()方法，通过调用 loadObject()方法对 xids 进行恢复。

```java
public static boolean storeObject(Object o, String path) {
    File xidLog = new File(path);
    ObjectOutputStream oout = null;
    try {
        oout = new ObjectOutputStream(new FileOutputStream(xidLog));
        oout.writeObject(o);
        oout.flush();
        return true;
    } catch (Exception e) {
        return false;
    } finally {
        try {
            if (oout != null)
                oout.close();
        } catch (IOException e1) {
        }
    }
}
```

```java
public static Object loadObject(String path) {
    File x = new File(path);
    ObjectInputStream oin = null;
    try {
        oin = new ObjectInputStream(new FileInputStream(x));
        return oin.readObject();
    } catch (Exception e) {
        return null;
    } finally {
        try {
            if (oin != null)
                oin.close();
        } catch (IOException e1) {
        }
    }
}
```

```java
private void recover() {
    Object xids_tmp = Util.loadObject(xidsLog);
    if (xids_tmp != null)
        xids = (HashSet<Integer>) xids_tmp;
}
```

当执行了 WorkFlowControllerImpl 中的 start()，abort()，commit()方法后，如上文所述，xids 中的事务号均发生了变化，因此我们在每个方法中调用 Util.storeObject()函数，更新磁盘所保存的 xids 对象。为了应对 WorkFlowControllerImpl 宕机的情况，在新建该类对象时，首先调用 recover()方法读取磁盘上保存的 xids，以此避免事务的丢失。

3.2 xid 异常

如前文所述，所有数据操作方法及 abort()与 commit()方法均需要指定事务号 xid 进行，表

示该事务调用了当前方法。因此，在所有方法的实现最初，都需在保存事务号的哈希表 xids 中查找当前 xid 是否存在。若当前指定的 xid 不存在，说明当前事务不存在（如该事务已被提交），则方法抛出非法事务异常。

```
if (!xids.contains(xid))
    throw new InvalidTransactionException(xid, "commit");
```

3.3 死锁异常

由于本项目通过使用事务锁处理冲突，在对不同数据表进行增删改查时可能出现死锁的问题，体现在代码中为调用 ResourceManager 的 query()，insert()，update()，delete()等方法时，ResourceManager 可能抛出 DeadLockException 异常，当 WorkFlowControllerImpl 收到某个 ResourceManager 抛出的 DeadLockException 时，会调用自己的 abort()方法终止当前事务，并再次抛出事务终止异常 TransactionAborted。