JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Verteilte Transaktionen

Systemtechnik Labor
5BHITT 2015/16, Gruppe Z

Mathias Ritter

Version 1.0

Note:

Betreuer: Prof. Micheler

Begonnen am 12. Februar 2016

Beendet am 14. Februar 2016

Inhaltsverzeichnis

1Einführung....3
1.1Ziele....3
1.2Voraussetzungen....3
1.3Aufgabenstellung....3
2Ergebnisse....5
2.1Kommunikation zwischen Transaktionsmanager und Stationen....5
Sockets....5
Aufbau einer Nachricht....5
2.2Ablauf der Kommunikation....6
2.3Funktionsweise des Transaktionsmanagers....7
2.4Funktionsweise der Stationen....8
2.5Starten der Anwendungen....8
Starten des Transaktionsmanagers....8
Starten der Stationen....9
2.6Beispielausführung....9
Ausgabe des Transaktionsmanagers....9
Ausgabe der Station....9
2.7Zeitaufwand & Probleme....9

1 Einführung

Die Übung soll die Grundlagen von verteilte Transaktionen mit Hilfe eines praktischen Beispiels in JAVA vertiefen.

1.1 Ziele

Implementieren Sie in JAVA einen Transaktionsmanager, der Befehle an mehrer Stationen weitergibt und diese koordiniert. Mit Hilfe des 2-Phase-Commit Protokolls sollen die Transaktionen und die Antwort der Stationen verwaltet werden. Der Befehl kann beliebig gewaehlt werden und soll eine Datenquelle (Datenbank oder Datei oder Message Queue) abfragen oder veändern.

Die Kommunikation zwischen Transaktionsmanagers und der Stationen soll mit Hilfe einer Übertragungsmethode (IPC, RPC, Java RMI, JMS, etc) aus dem letzten Schuljahr umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen Transaktionen (allgmein, Datenbanksysteme)
- Anbindung Datenquelle in JAVA (JDBC, File, JMS)
- Kommunikation in JAVA (IPC, RPC, Java RMI, JMS)

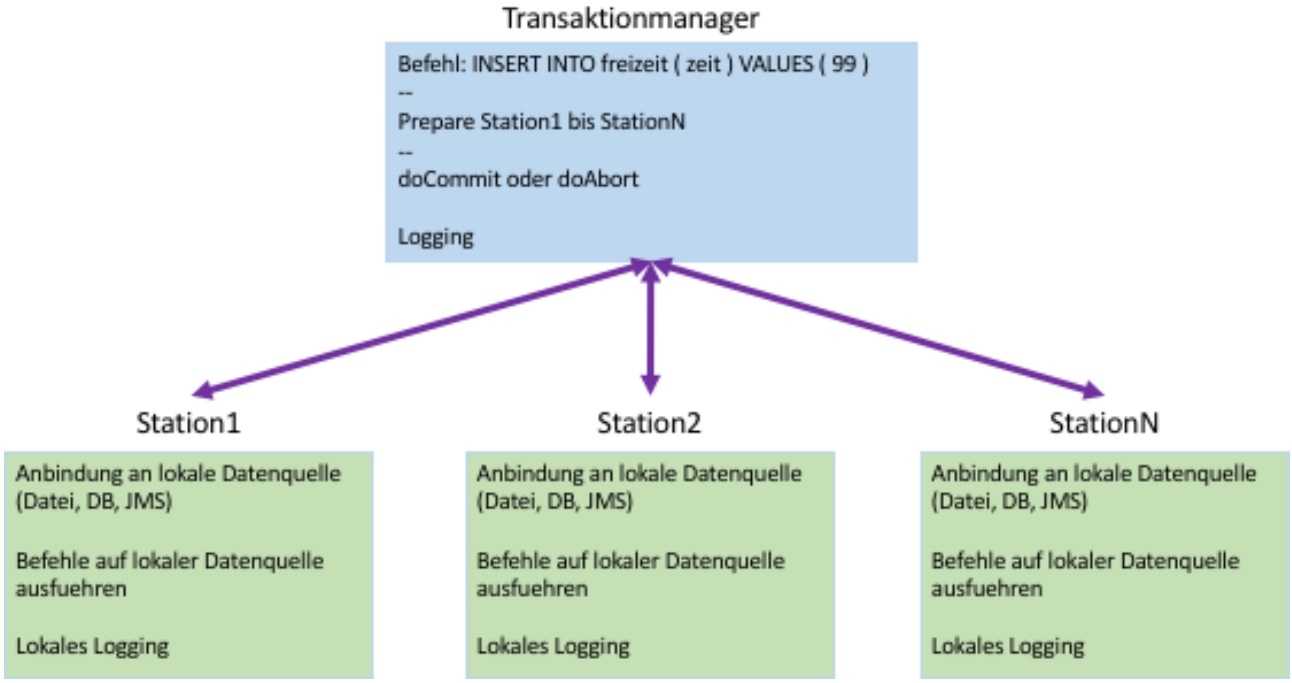
1.3 Aufgabenstellung

Der Transaktionsmanager läuft auf einer eigenen Instanz (bzw. eigenem Port) und stellt die Schnittstelle zwischen den Stationen und dem Benutzer dar. Der Benutzer gibt über die Konsole oder ein User Interface einen Befehl ein, der danach an alle Stationen verteilt wird. Da das 2-Phase-Commit Protokoll als Transaktionsprotokoll zugrunde liegt, soll der Transaktionsmanager jeweils nach einem Befehl,

das Resultat nach der PREPARE-Phase (Bsp. 3xYES 0xNO 0xTIMEOUT) ausgeben
welche Aktion der Transaktionsmanager danach durchfuehrt (doCommit, doAbort)
das Resultat der COMMIT-Phase (Bsp. 3xACK 0xNCK 0xTIMEOUT)
danach kann ein neuer Befehl eingegeben werden
Logging:
Um im Einzelfall die Transaktionen und Resultat nachverfolgen zu koennen, sollen alle Befehle und deren Resultate mit Timestamp geloggt werden. Beim Transaktionsmanager soll dokumentiert werden, welcher Befehl zu welcher Station und zu welchem Zeitpunkt abgesendet wurde, ebenso beim Erhalt der

Antwort. Ebenso sollen, bei den Stationen eingehenden Befehle und deren Resultate bei Ausführung an der lokalen Datenquelle mitdokumentiert werden.

Die folgende Grafik soll den Vorgang beim 2-Phase-Commit Protokoll verdeutlichen:



Bewertung: 16 Punkte
- verteilte Transaktion mit 1 Station (8 Punkte)
- verteilte Transaktion mit n Stationen (4 Punkte)
- Logging (in Log-Dateien) Transaktionsmanager und Stationen (2 Punkte)
- Protokoll (2 Punkte)

Alternativ zu dieser Aufgabenstellung kann ein Transaktionssystem in JEE mit Hilfe der JTA entwickelt werden. Details mit Prof. Micheler per Mail abklären.

2 Ergebnisse

2.1 Kommunikation zwischen Transaktionsmanager und Stationen

Sockets

Die Kommunikation zwischen Transaktionsmanager und den Stationen erfolgt über Sockets. Dafür wurden die Klassen `SocketBase`, `SocketServer` und `SocketClient` implementiert.

Die abstrakte Klasse `SocketBase` enthält Methoden, welche sowohl vom Client als auch vom Server benötigt werden. Dazu gehört das Senden und Empfangen von Nachrichten und das Beenden einer Verbindung. Die Methode zum Aufbauen zur Verbindung ist abstrakt, da eine unterschiedliche Implementierung bei Server und Client notwendig ist.

Beim Senden einer Nachricht wird zuerst der Typ der Nachricht, danach die Länge und schlussendlich der Inhalt übertragen (siehe „Aufbau einer Nachricht“ weiter unten).

```
out.writeChar(message.getType().getValue());
out.writeInt(message.getLength());
out.write(message.getContent());
```

Beim Empfangen werden diese Parameter in der selben Reihenfolge abgerufen. Das Empfangen von Nachrichten erfolgt in einer Schleife in der Run-Methode. Wird eine neue Nachricht empfangen, so wird anschließend eine Methode der Klasse aufgerufen, die die Nachricht weiterverarbeitet.

Aufbau einer Nachricht

Die Klasse Message stellt den Aufbau einer Nachricht dar. Sie enthält einen `MessageType`, welcher den Typ einer Nachricht angibt. Dieser ist notwendig, damit Transaktionsmanager und Stationen unterscheiden können, wie die zu empfangene Nachricht weiterzuverarbeiten ist. Folgende Nachrichtentypen werden benötigt und wurden implementiert:

```
CLIENT_CONNECTED('E'), SQL_QUERY('Q'), DO_COMMIT('C'), DO_ABORT('A'),
RESULT('R');
```

Weiters enthält sie ein Attribut `content`, welches den Inhalt der Nachricht als Byte-Array enthält.

2.2 Ablauf der Kommunikation

Die Kommunikation zwischen dem Transaktionsmanager und den Stationen läuft folgendermaßen ab:

1. Starten des Transaktionsmanager: Dieser akzeptiert neue Verbindungen durch n Stationen.

2. Starten einer oder mehrerer Stationen: Diese melden sich beim Transaktionsmanager an. Dabei senden sie eine Nachricht vom Typ `LIENT_CONNECTED`.

3. Der Transaktionsmanager erhält die Nachricht und erstellt für die Station einen neuen Thread, der für die Kommunikation mit der Station zuständig ist.

4. Der Benutzer gibt einen Befehl ein (INSERT/UPDATE/DELETE). Der Transaktionsmanager sendet den Befehl an alle Stationen. Es wird eine maximale Antwortzeit festgelegt.

5. Die Stationen antworten entweder mit YES oder NO. Falls eine Station nicht rechtzeitig antwortet, wird dies als TIMEOUT gezählt.

6. Der Transaktionsmanager entscheidet, ob ein COMMIT oder ein ABORT erfolgt. Ein COMMIT erfolgt nur, wenn alle Stationen mit YES antworteten. Der Transaktionsmanager sendet COMMIT oder ABORT an alle Stationen. Es wird eine maximale Antwortzeit festgelegt.

7. Die Stationen antworten entweder mit ACK oder mit NCK. Falls eine Station nicht rechtzeitig antwortet, wird dies als TIMEOUT gezählt.

2.3 Funktionsweise des Transaktionsmanagers

Nach dem Starten des Transaktionsmanagers wartet dieser auf eine Benutzereingabe. Gibt der Benutzer ein SQL-Statement ein, so wird dieses im ersten Schritt an alle Stationen gesendet. Ab nun werden vorerst keine weiteren Benutzereingaben verarbeitet. Außerdem wird das Timeout auf die aktuelle Zeit plus die maximale Antwortzeit (1000ms) gesetzt.

```
this.timeout = System.currentTimeMillis() + maxTime;
this.socket.write(new Message(input, MessageType.SQL_QUERY));
```

Der Transaktionsmanager wartet auf die Antworten der Stationen. Je nach Antwort wird der Counter für die erfolgreichen und nicht erfolgreichen Antworten erhöht. Am Ende der Verarbeitung jeder Antwort wird überprüft, ob bereits alle Stationen antworteten:

```
if (this.requiredResponses == this.successResponses + this.failResponses)
{
this.checkResponses();
}
```

Außerdem werden die erhaltenen Antworten auf per Aufruf der selben Methode nach Ablauf des Timeouts geprüft. Sind bei der Überprüfung NO-Antworten oder TIMEOUTs vorhanden, so sendet der Manager ein ABORT an alle Stationen, sonst ein COMMIT:

```
if (this.failResponses > 0 || timeoutResponses > 0) {
      this.socket.write(new Message("", MessageType.DO_ABORT));
} else {
      this.socket.write(new Message("", MessageType.DO_COMMIT));
}
```

Das Timeout wird wieder auf die aktuelle Zeit plus die maximale Antwortzeit (1000ms) gesetzt. Die Counter für die erhaltenen Antworten werden zurückgesetzt. Nach erhalten aller erforderlichen Antworten bzw. nach Ablauf des Timeouts wird das Ergebnis in der Konsole ausgegeben und es wird auf die nächste Benutzereingabe gewartet.

2.4 Funktionsweise der Stationen

Nach dem Starten einer Station wird eine Datenbankverbindung aufgebaut. Momentan ist der Verbindungsaufbau nur für MySQL implementiert, allerdings lässt sich die Anwendung einfach erweitern, um auch andere DBMS zu unterstützen.

Danach wartet die Station auf Befehle des Transaktionsmanagers. Zuerst wird ein SQL-Statement empfangen, welches ausgeführt wird. Je nachdem, ob das Ausführen der Anweisung erfolgreich war oder nicht, wird YES oder NO zurück an den Transaktionsmanager gesendet. Die Methode zur Ausführung des SQL-Befehls liefert deshalb true oder false zurück.

```
if (sqlStatement.toUpperCase().startsWith("INSERT") ||
            sqlStatement.toUpperCase().startsWith("UPDATE")) {
      this.connection.prepareStatement(sqlStatement).executeUpdate();
      success = true;
} else {
      success = this.connection.prepareStatement(sqlStatement).execute();
}
```

Danach wartet die Station auf den COMMIT- oder ABORT-Befehl des Transaktionsmanagers. Im Fall von COMMIT wird dieser durchgeführt, im Fall von ABORT wird ein Rollback der Transaktion durchgeführt:

```
if (commit)
      this.connection.commit();
else
      this.connection.rollback();
```

Wurde der Befehl erfolgreich durchgeführt (ist keine Exception aufgetreten), wird wieder true zurückgegeben. Die Station sendet ACK an den Transaktionsmanager, sonst sendet diese NCK.

2.5 Starten der Anwendungen

Starten des Transaktionsmanagers

Um den Transaktionsmanager zu starten, führt man in der Konsole die Jar-Datei „dezsys06_manager.jar“ aus. Dabei kann man folgende Parameter angeben:

- „P“: Port des Servers, auf den sich die Stationen verbinden können. Default: 35768

Starten der Stationen

Um die Station zu starten, führt man in der Konsole die Jar-Datei „dezsys06_station.jar" aus. Dabei kann man folgende Parameter angeben:

- „h": IP-Adresse des Transaktionsmanagers. Default: 127.0.0.1
- „P": Port des Transaktionsmanagers. Default: 35768
- „u": Username des DBMS. Default: Angemeldeter Benutzer
- „p": Passwort des DBMS. Default: keines
- „**d**": Datenbank des DBMS.

2.6 Beispielausführung

Ausführen eines Inserts durch Eingabe beim Transaktionsmanager mit einer Station:

Ausgabe des Transaktionsmanagers

```
2016-02-14 19:38:55,244 INFO - Received CLIENT_CONNECTED
insert into freizeit(zeit) values (1);
2016-02-14 19:39:11,858 INFO - Received RESULT
2016-02-14 19:39:11,859 INFO - Result is YES
2016-02-14 19:39:11,859 INFO - YES: 1, NO: 0, TIMEOUT: 0
2016-02-14 19:39:11,860 INFO - Received RESULT
2016-02-14 19:39:11,860 INFO - Result is ACK
2016-02-14 19:39:11,860 INFO - ACK: 1, NCK: 0, TIMEOUT: 0
```

Ausgabe der Station

```
2016-02-14 19:39:11,837 INFO - Received SQL_QUERY
2016-02-14 19:39:11,838 INFO - Executing statement: insert into
freizeit(zeit) values (1);
2016-02-14 19:39:11,858 INFO - Successfully ecexuted statement.
2016-02-14 19:39:11,859 INFO - Received DO_COMMIT
2016-02-14 19:39:11,859 INFO  - Ending transaction with commit
2016-02-14 19:39:11,860 INFO  - Successfully ended with commit.
```

2.7 Zeitaufwand & Probleme

Bei der Implementierung sind keine gröberen Probleme aufgetreten. Ich benötigte für diese Aufgabe insgesamt ca. 8 Stunden.