Содержание

Содержание 1
Основные положения технического задания и их решение 2
Архитектура решения 2
Структура проекта 4
Сборка, настройка и тестирование 4
Конфигурация RA и MDB 8
Ограничения и to-do list 8

Основные положения технического задания и их решение

- *Обработка элементов внутри группы выполняется строго последовательно:* выборка элементов из базы выполняется с условием ORDER BY, выгруженные элементы передаются приложению на обработку в структуре, которая гарантирует порядок записей, сформированный на этапе заполнения структуры;
- *Группы могут обрабатываться параллельно*: реализовано за счет работы нескольких потоков выборки данных их базы и пула приложений для финальной обработки групп;
- *Требуется простейшая обработка каждого элемента группы на уровне получателя сообщения*: по каждому элементу считается статистика, которая при необходимости выводится в лог;
- *Элементы в исходную таблицу добавляются асинхронно и параллельно с обработкой:* реализовано с помощью отдельного приложения DB-GEN;
- *После обработки элемент должен быть удален из исходной таблицы*: реализовано при извлечении элементов из базы в пределах текущей транзакции;
- *Необходима равномерная обработка групп*: группы из очереди выбираются в случайном порядке;
- *Приложение обработки сообщений не должно завершать свою работу при отсутствии элементов в исходной таблице*: реализовано за счет функционала сервера приложений.

Архитектура решения

Техническое задание реализовано в виде набора модулей для сервера приложений GlassFish, структура модулей показана на «Рисунке 1». Зеленым цветом указаны самостоятельно разработанные компоненты, белым цветом – внешние библиотеки и компоненты сервера приложений. Серым цветом отмечены функциональные блоки системы.

Логически предлагаемую реализацию можно разделить на две части:

1. Внешняя информационная система, накапливающая очередь сообщений в базе данных (на «Рисунке 1» обозначена как EIS).
2. Сервер приложений, на базе которого реализован функционал выборки и обработки очереди.

Для построения тестового стенда внешней информационной системы была выбрана база PostgreSQL, запись сообщений в базу осуществляется приложением DB-GEN. Указанное приложение собирается средствами Maven, в реализации используется простое API на основе Hibernate, очередь сообщений в базу формируется в нескольких JUNIT тест кейсах.

Основная часть технического задания реализована в виде адаптера ресурсов (inbound flow only) для сервера приложений GlassFish, а также MDB бина, предназначенного для окончательной обработки входящих сообщений из очереди (таблицы) внешней информационной системы.

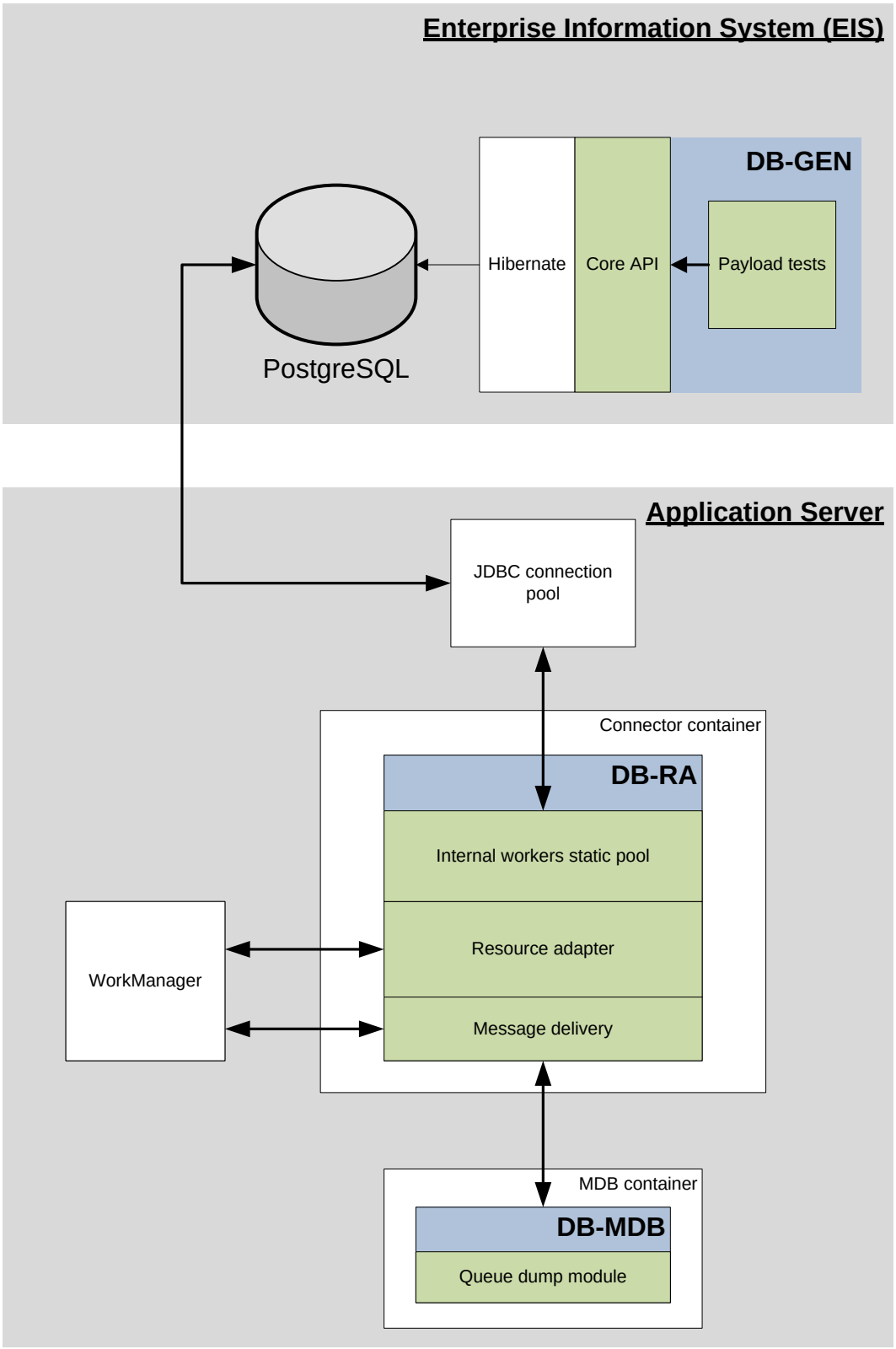

Рисунок 1. Общая схема реализации тестового задания.

Так как вывод на печать большого количества входящих сообщений (согласно ТЗ) сложно анализировать, в MDB был реализован простой анализ входящих групп и их записей с выводом финального дампа с консолидированной информацией по всем принятым и обработанным сообщениям. Все входящие группы записей обрабатываются и проверяются следующим образом:

- Идентификаторы записей в группе должны быть получены (и соответственно обработаны в MDB) строго последовательно (каждая последующая запись в группе должна иметь больший идентификатор, чем предыдущая запись). *Допускается последующий прием группы с таким же именем, но с меньшими значениями идентификаторов записи, так как такая последовательность может быть записана на стороне EIS*;
- Группа должна содержать хотя бы одну запись;
- Подсчитываются все обработанные сообщения в каждой группе;
- Подсчитывается разница во времени между получением нескольких групп с одним и тем же именем;
- Подсчитывается время между обработкой первой и последней группы, по этим значениям вычисляется TPS (среднее количество обработанных записей во всех группах за все время обработки);

Статистика накапливается в MDB до получения группы с заранее предопределенным именем (в текущей реализации – группа с именем «AUDIT», параметр конфигурируется в настройках MDB). При получении такой записи MDB выводит накопленную статистику (финальный дамп) в лог сервера приложений ~~и умирает~~. При этом структура с данными последнего аудита очищается для того, чтобы можно было повторить тестирование системы с любым другим набором данных.

Структура проекта

- db-gen – Maven проект с реализацией эмулятора EIS, обеспечивает запись тестовых элементов в базу PostgreSQL;
- doc – каталог с документацией;
- db-mdb – Maven проект MDB бина, обрабатывает (консолидирует) сообщения из очереди в базе данных;
- db-ra – Maven parent POM проект, обеспечивает общую конфигурацию для всех проектов;
- db-ra-api– Проект для общих классов, используемых в приложениях db-mdb и db-ra-core;
- db-ra-rar - Maven проект адаптера ресурсов, содержит только конфигурацию адаптера и правила для его сборки.

Сборка, настройка и тестирование

Для сборки, настройки и тестирования рекомендуется использовать Netbeans 7.3 и GlassFish 3.1.2.2.

Создаем тестовую базу данных. Для этого подключаемся к PostgreSQL с праваими администратора и создаем пользователя testdbra с паролем testdbra:

```
CREATE ROLE testdbra LOGIN
```

```
  ENCRYPTED PASSWORD 'md51d349a7e6b868296c2c70316398fdcba'
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE;
```

Создаем базу testdbra:

```
CREATE DATABASE testdbra
  WITH OWNER = testdbra
       ENCODING = 'UTF8'
       TABLESPACE = pg_default
       LC_COLLATE = 'Russian_Russia.1251'
       LC_CTYPE = 'Russian_Russia.1251'
       CONNECTION LIMIT = -1;
```

На этом этапе необходимо отключиться от PostgreSQL (так как сейчас у нас права администратора) и подключиться к базе testdbra, используя логин и пароль testdbra. Создаем таблицу для хранения очереди:

```
CREATE TABLE public.message
(
  id bigint NOT NULL,
  item_id bigint NOT NULL,
  group_id text NOT NULL,
  message text,
  CONSTRAINT id_pk PRIMARY KEY (id),
  CONSTRAINT item_id_un UNIQUE (item_id)
)
WITH (
  OIDS=FALSE
);
```

Создаем счетчик для индекса, без него epic fail hibernate слоя обеспечен на 100%:

```
CREATE SEQUENCE seq_id
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
```

На этом квест с созданием базы можно считать завершенным, переходим к танцам с сервером приложений.

Запускаем GlassFish из командной строки (или через Netbeans):

```
C:\services\glassfish-3.1.2.2\glassfish\bin>asadmin.bat start-domain
Waiting for domain1 to start .............
Successfully started the domain : domain1
domain  Location: C:\services\glassfish-3.1.2.2\glassfish\domains\domain1
Log File: C:\services\glassfish-3.1.2.2\glassfish\domains\domain1\logs\server.log
Admin Port: 4848
Debugging is enabled.  The debugging port is: 9009
Command start-domain executed successfully.
```

Создаем файл с паролем администратора, чтобы GlassFish не спрашивал пароль для каждой операции:

```
C:\services\glassfish-3.1.2.2\glassfish\bin>asadmin login --host localhost --port 4848
Deprecated syntax, instead use:
asadmin --host localhost --port 4848 login [options] ...
Enter admin user name [default: admin]>
Admin login information for host [localhost] and port [4848]
is being overwritten with credentials provided because the
--savelogin option was used during the create-domain command.
Login information relevant to admin user name [admin]
for host [localhost] and admin port [4848] stored at
[C:\Users\rk\.asadminpass] successfully.
Make sure that this file remains protected.
Information stored in this file will be used by
asadmin commands to manage the associated domain.
Command login executed successfully.
```

Создаем connection pool ск базе данных (с именем dbPoolNoXA, без поддержки XA), адаптер ресурсов будет подключаться к базе через это соединение:

```
C:\services\glassfish-3.1.2.2\glassfish\bin\asadmin.bat create-jdbc-connection-pool
--user admin
--datasourceclassname org.postgresql.ds.PGPoolingDataSource
--restype javax.sql.ConnectionPoolDataSource
--property
user=testdbra:password=testdbra:url=\"jdbc:postgresql://localhost:5432\":databaseName=testdbra
--allownoncomponentcallers=true dbPoolNoXA
JDBC connection pool dbPoolNoXA created successfully.
Command create-jdbc-connection-pool executed successfully.
```

Создаем JDBC соединение для созданного пула:

```
C:\services\glassfish-3.1.2.2\glassfish\bin\asadmin.bat create-jdbc-resource
--user admin --connectionpoolid dbPoolNoXA --enabled=true jdbc/dbPoolNoXA
JDBC resource jdbc/dbPoolNoXA created successfully.
Command create-jdbc-resource executed successfully.
```

Собираем адаптер и MDB, для этого выбираем для проектов db-ra и db-ra-core конфигурацию glassfish-embedded-3.1 и собираем родительский POM проект db-ra, все зависимые проекты (кроме DB-GEN) соберутся автоматически.

Устанавливаем адаптер ресурсов в сервер приложений:

```
move .\db-ra-rar\target\db-ra-rar-1.0-SNAPSHOT.rar db-ra.rar
C:\services\glassfish-3.1.2.2\glassfish\bin\asadmin.bat deploy --user admin db-ra.rar
Application deployed with name db-ra.
Command deploy executed successfully.
```

Внимание, для корректной работы необходимо устанавливать файл адаптера с именем db-ra.rar, если устанавливать файл с именем **db-ra-rar-1.0-SNAPSHOT.rar**, то нужно изменить имя адаптера в дескрипторе MDB для корректной работы приложений.

Устанавливаем MDB в сервер приложений:

```
C:\services\glassfish-3.1.2.2\glassfish\bin\asadmin.bat deploy --user admin db-
mdb/target/db-mdb-1.0-SNAPSHOT.jar
Application deployed with name db-mdb-1.0-SNAPSHOT.
Command deploy executed successfully.
```

Собираем проект DB-GEN, в процессе сборки проекта будут запущены тесты, которые добавят в таблицу базы данных некоторое количество тестовых записей двумя алгоритмами (линейный тест и тест с добавлением случайных наборов групп). Если все пройдет успешно, в логе сборки проекта должны появиться следующие записи:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running ru.nsk.test.db.gen.CoreTest
0    [main] INFO  ru.nsk.test.db.gen.Core  - Static initializer called.
Init+Destroy
MergeMessage
914    [main] INFO   ru.nsk.test.db.gen.CoreTest   - Saved object Message: ID=100 104,
GROUP=TEST, ITEM=1, TEXT=Test record
914  [main] INFO  ru.nsk.test.db.gen.CoreTest  - Saved object ID 100 104
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.108 sec
Running ru.nsk.test.db.gen.PayloadLinearTest
testPayloadLinear
1131 [main] INFO   ru.nsk.test.db.gen.PayloadLinearTest   - Test waiting while table for
entity Message has no records.
1131 [main] INFO  ru.nsk.test.db.gen.PayloadLinearTest  - If test freeze too long, please
check RA+MDB in application server...
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.173 sec
Running ru.nsk.test.db.gen.PayloadRandomTest
```

```
testPayloadRandom
91075 [main] INFO  ru.nsk.test.db.gen.PayloadRandomTest  - Stored 100 000 items in 87 seconds, TPS 1 149.
91075 [main] INFO  ru.nsk.test.db.gen.PayloadRandomTest  - Test waiting while table for entity Message has no records.
91075 [main] INFO  ru.nsk.test.db.gen.PayloadRandomTest  - If test freeze too long, please check RA+MDB in application server...
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 119.148 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

Тест PayloadLinearTest добавляет в очередь одну группу, содержащую 100 записей. После того, как группа будет обработана сервером приложений, тест добавляет 1 группу с именем «AUDIT», что служит для MDB сигналом к выводу статистики. В логе сервера приложений должны появиться следующие записи:

```
AUDIT -> Processed group LinearTest in 1 parts, total 100 items, group elapse 0 msec.|#]
AUDIT -> total in all groups: 100 items, processed in 0 sec, TPS ?.|#]
```

Тест PayloadRandomTest формирует 1000 групп по 100 элементов, после чего в случайном порядке записывает элементы в очередь базы данных. Таким образом, в базе появляются группы со случайным количеством записей, в случайном порядке, причем группы с одним и тем же именем могут быть записаны повторно (с соблюдением требования об уникальности номеров элементов). После выгрузки всех записей в базу тест ожидает, пока MDB вычитает все элементы, затем тест записывает одну запись группы «AUDIT». При получении такой записи MDB записывает в лог сервера приложений статистику, которая может выглядеть следующим образом:

```
…
AUDIT -> Processed group RandomBlock-228 in 6 parts, total 100 items, group elapse 111 696 msec.|#]

AUDIT -> Processed group RandomBlock-434 in 4 parts, total 100 items, group elapse 134 385 msec.|#]

AUDIT -> Processed group RandomBlock-435 in 4 parts, total 100 items, group elapse 121 844 msec.|#]

AUDIT -> Processed group RandomBlock-432 in 6 parts, total 100 items, group elapse 137 952 msec.|#]

AUDIT -> Processed group RandomBlock-433 in 5 parts, total 100 items, group elapse 123 227 msec.|#]

AUDIT -> Processed group RandomBlock-438 in 5 parts, total 100 items, group elapse 131 281 msec.|#]

AUDIT -> Processed group RandomBlock-439 in 5 parts, total 100 items, group elapse 124 842 msec.|#]

AUDIT -> Processed group RandomBlock-436 in 3 parts, total 100 items, group elapse 118 055 msec.|#]

AUDIT -> Processed group RandomBlock-437 in 6 parts, total 100 items, group elapse 138 385 msec.|#]

AUDIT -> total in all groups: 100 000 items, processed in 143 sec, TPS 699,301.|#]
```

Количество элементов в каждой группе (и в итоге) должно соответствовать исходным значениям (100 элементов в группе, итого 100 000 элементов). Если в результате работы тестов мы получаем именно такие результаты, то это дает нам основание полагать с некоторой (не очень большой степенью) вероятности, что все сообщения доставлены корректно, ничего не потеряно и не обработано дважды.

Конфигурация RA и MDB

Основные параметры приложений вынесены в дескрипторы адаптера и MDB бина.

Параметры адаптера (файл ra.xml в проекте db-ra-rar):

- poolSize – определяет количество потоков для мониторинга базы данных. Используется статический пул (смотрите Ограничения и to-do list), т.е. указанное количество потоков запускаются одновременно и существуют все время активной жизни адаптера.

Параметры MDB бина (файл ejb-jar.xml):

- resource – указывает на JNDI имя соединения к базе данных EIS;
- table – название таблицы, в которой хранится очередь EIS;
- fieldGroup – название поля таблицы, в котором хранится имя группы;
- groupLimit – количество групп, которое извлекается одним потоком за один цикл мониторинга таблицы EIS;
- fieldItem – название поля таблицы, в котором хранится идентификатор элемента группы;
- fieldMessage – название поля таблицы, в котором хранится текстовое сообщение элемента группы;
- command.audit – Название группы, которая для MDB является сигналом записи консолидированной статистики в лог сервера приложений.

Ограничения и to-do list

По причине ограниченного объема времени, выделенного на реализацию тестового задания, не реализованы следующие плюшки:

- Поддержка распределенных транзакций. При падении приложения группа, выгруженная из очереди, но не обработанная в MDB - будет потеряна;
- Простая реализация статического пула потоков для мониторинга новых элементов в таблице базы данных. Предполагается, что такая реализация достаточна для тестового задания. При разработке адаптера в практических целях необходимо создать динамический расширяемый пул потоков для мониторинга базы.
- Простая синхронизация потоков – «обработчиков групп» через статический HashMap достаточна при работе адаптера на одном узле. При работе адаптера в кластере такая реализация бесполезна, может привести к лишней нагрузке на CPU и блокировкам на уровне базы. Необходимо вынести синхронизацию обработки групп на внешний кластерный ресурс (например, на отдельную таблицу в той же базе).
- В силу того, что не удалось с ходу запустить Arquilian контейнер адаптера для тестирования решения в J-Unit тестах, указанные тесты не реализованы. Стабильность работы проверена только на функциональных тестах, выполненных через приложение DB-GEN.