Ответы на вопросы

Данис Тазеев

По «плюсам»

4. Присутствует документация

Какая документация? JavaDoc? Ну, да, кое-где есть JavaDoc, комментарии к коду встречаются чаще :).

7. Отделена бизнес логика от GUI

Вы имеете в виду, что запросы вынесены в DAO? Если именно это Вы имеете в виду, то да, отделена.

По «минусам»

1. дублирование кода: например, DateRenderer

Совершенно с Вами согласен. В предыдущем письме я писал, что сделал ревью кода. Общую часть классов `DateRenderer` из `DailyReportPanel` и `HistoryPanel` вынес в отдельный класс `CustomDateRenderer`. Теперь `HistoryPanel` использует прямо его, а `DailyReportPanel` — его наследника, внутренний класс `DateRenderer`.

2. странный запрос «update attendance set checked_out = ? where employee_id = ? and id = (select id from attendance where employee_id = ? and checked_out is null)». Можно упросить

Да, действительно странный. Вот упрщённый:

```
update attendance set checked_out = ?
where employee_id = ? and checked_out is null.
```

Однако! Такой запрос всё же отличается от оригинального. В оригинальном запросе возникает ошибка, если количество строк, возвращаемых подзапросом, больше одной. Такая ситуация означает, что пользователь AttendanceDAO не выполнил контракт операций `checkIn`/`checkOut`. Контракт такой: за каждым `checkIn` должен следовать `checkOut` (нельзя вызывать `checkIn` после `checkIn`), а `checkIn` не должен выполняться, если есть открытые интервалы.

Оригинальный запрос тоже не до конца проверяет контракт: если подзапрос возвращает 0 строк, то `update` не обновляет ни одной строки. Что тоже является нарушением (`checkOut` вызван после `checkOut`).

В любом случае, в коде выполнена проверка только половины контракта и то не до конца: проверяется, что за каждым `checkIn` следует `checkOut`, но не проверяется, что `checkIn` не открывает новый интервал, если есть уже открытый. Также не проверяется, что для `checkOut` открытый интервал есть. Проверятся только, что их не больше одного.

С упрощённым запросом проверку `checkOut` после `checkIn` тоже можно выполнить, проверив, что `PreparedStatement.executeUpdate()` возвращает 1. Правда, в этом случае придётся выключать режим auto-commit и откатывать транзакцию, если `executeUpdate()` вернул не 1.

Преверять, что `checkIn` вызывается при отсутствии открытых интервалов без дополнительного запроса невозможно. Я бы не стал вводить новый запрос — это всё-таки запрос, значит, не только время и ресурсы СУБД на его выполнение, но и сетевой round-trip (если СУБД удалённая).

В итоге согласен упрстить оригинальный запрос, выключить auto-commit и откатывать транзакцию, если открытых интервалов для `checkOut` нет. Проверять, что открытых интервалов для `checkIn` нет, я бы просто не стал из-за дополнительного запроса.

3. Использование static вместо шаблона Singleton: например, «static final Connection»

А в чём, собственно, проблема? Очень уместно использовать единственное соединение для всех DAO. Не согласен с минусом.

5. Зачем что плодятся экземпляры AttendanceDAO - ведь кажется объект без состояния? Опять же почему не Singleton?

DAO — объект с состоянием. Вот его состояние:

- флажок `closed`, используемый для гарантирования использования соединения только каким-либо одним экземляром DAO;
- кеш `PreparedStatement`. Может показаться, что он лишний. В данной программе, да, лишний. Но в общем случае очень даже нужный. Дело в том, что какой-нибудь алгоритм может вызывать одну и ту же операцию конкретного DAO в цикле. Тогда не придётся каждый раз создавть `PreparedStatement` — делать round-trip к СУБД;
- `lastSql`: используется для логирования возникающих сбоев или ошибок. Незаменим при отладке и даже эксплуатации.

Не согласен с минусом.

6. Не понравилась реализация многопоточности

Что именно не понравилось и какой именно многопоточности? Приложение-то в основном однопоточное. Это же не сервер, обслуживающий одновременно нескольких пользователей (клинтов). Многопоточность реализована корректно. Не согласен с минусом. Не понимаю, что именно Вам не понравилось.

Ответы на вопросы

1. Как бы Вы изменили приложение, если бы встала задача поддержки нескольких типов БД?

Я бы каждый DAO (каждый конкретный наследник абстрактного `DAO`) конфигурировал поставщиком запросов. Это один вариант, который может и не сработать, если запросы отличаются количеством или последовательностью параметров.

Второй вариант — интерффейс для каждого DAO, объявляющий, какие операции, этот DAO может выполнять. И столько реализаций этого интерфейса, столько СУБД.

2. Как бы Вы изменили приложение, если бы встала задача разделения клиента и сервера приложения?

Собстенно, UI и DAO уже разделены. Отдельной логики в приложении нет. Так что я бы просто вынес DAO в отдельный процесс, обернул бы их либо в EJB, либо в Spring-сервисы, и всё.

3. Что Вы понимает под «вложенными транзакциями» и какая СУБД их поддерживает - я не знаю J? Или имеется в виду «вложенные транзакции» по отношению к приложению в целом?

Термин «вложенные транзакции» относится не к СУБД. Вот пример. Чтобы добавлять строчки в таблицы, нужна последовательность суррогатный идентификаторов. То, что в Oracle называется sequence.

Чтобы два раза не ходить в СУБД, делаем специальный сервис, который в БД выделяет идентификаторы блоками, и последовательно возвращает по одному из блока. Выделение нового блока в БД — отдельная транзакция.

```
class Sequencer {
  long nextValue(String sequenceId) { … }
}
```

Пусть есть метод, который вставляет строчку в БД, и выполняется в рамках транзакции (отличной от той, которая выделяет блок идентификаторов):

```
void addEmployee(String name) {
   long id = sequencer.nextValue(“employee”);
   try (EmployeeDAO dao = new EmployeeDAO()) {
      dao.insertEmployee(id, name);
   }
}
```

Смотрите, `addEmployee()` выполняется в какой-то транзакции. И если она сломается, то будет откачена. В то же время `addEmployee()` вызывает `Sequencer.nextValue()`. Если бы `nextValue()` выполнялся в той же транзакции, что и `addEmployee()`, то при откате транзакции, в рамках которой выполняется `addEmployee()`, откатилось бы и изменение,

сделанное `nextValue()`, то есть откатиось бы выделение блока идентификаторов, чего быть не должно. Это и есть вложенная транзакция.

Такой привет не единственный.

4. Как реализуется «очередь заданий», описываемая в doc? Не нашел в приложении. И если есть почему, он не выделен в отдельный компонент?

Очередь заданий — внутренности SwingWorker. См. документ «Механика SwingWorker».