

№ 14 Многоуровневая архитектура

Задание

Измените архитектуру предыдущей лабораторной (на основе 11). Архитектурно приложение должно быть многоуровневым (MVC) со слоем репозитория, сервисов (команд), представлений (jsp) и одним сервлетом (Controller) принимающим все запросы.

Для каждого уровня создайте свои классы исключений.

Используйте логгирование.

Напишите и добавьте класс пула соединений для базы данных. Используйте его в проекте.

При необходимости реализовать паттерны команда, абстрактная фабрика, строитель и синглтон.

Создайте два unit теста (используйте mock для зависимых объектов).

Вопросы

1. Для чего используется репозиторий, какие функции он выполняет?
2. Каково назначение слоя сервисов?
3. Расскажите что такое паттерн FrontController.
4. Как используется абстрактная фабрика в вашем проекте?
5. Расскажите о взаимодействии jsp-servlet-jsp.

Методические указания к выполнению работы.

ЛОГГИРОВАНИЕ

В программировании принято логировать практически все. Java-программы – это очень часто большие серверные приложения без UI, консоли и т.д. Они обрабатывают одновременно запросы тысяч пользователей, и нередко при этом возникают различные ошибки. Особенно, когда разные нити начинают друг другу мешать.

Фактически, единственным способом поиска редко воспроизводимых ошибок и сбоев в такой ситуации есть запись в лог/файл всего, что происходит в каждой нити.

Чаще всего в лог пишется информация о параметрах метода, с которыми он был вызван, все перехваченные ошибки, и еще много промежуточной информации.

Чем полнее лог, тем легче восстановить последовательность событий и отследить причины возникновения сбоя или ошибки.

Изначально в Java не было своего логгера, что привело к написанию нескольких независимых логгеров. Самым распространенным из них стал log4j.

Создается логгер следующей строкой в классе.

```
private static final Logger LOGGER = Logger.getLogger(ConnectionPool.class);
```

getLogger – это его статический метод. В него обычно передают текущий класс. Такой статический объект создают практически в каждом классе.

Обычно, у каждого лог-сообщения есть своя степень важности, и, используя ее можно часть этих сообщений отбрасывать. Степени важности:

Степень важности	Описание
ALL	Все сообщения
TRACE	Мелкое сообщение при отладке
DEBUG	Сообщения важные при отладке
INFO	Просто сообщение
WARN	Предупреждение
ERROR	Ошибка

FATAL	Фатальная ошибка
OFF	Нет сообщения

Если выставить уровень логирования в WARN, то все сообщения, менее важные, чем WARN будут отброшены: TRACE, DEBUG, INFO.

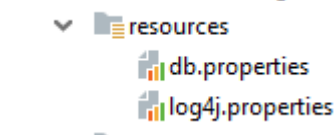
Если выставить уровень фильтрации в FATAL, то будут отброшены даже ERROR'ы.

Есть еще два уровня важности, которые используются при фильтрации – это OFF – отбросить все сообщения и ALL – показать все сообщения (не отбрасывать ничего).

Пример записи в лог

```
LOGGER.error("Can not get Instance", e);
```

Настройки логгера log4j задаются в файле log4j.properties.



```
# Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n

# Redirect log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
#outputs to home
log4j.appender.file.File=log/applicationLog
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n
```

В этом файле можно задать несколько appender'ов – объектов, в которые будут писаться данные. Есть источники данных, а есть – аппендеры – объекты, куда как бы «стекают» данные.

Мы указываем уровень сообщений, которые оставляем. Все менее важные уровни будут отброшены (DEBUG). Там же, через запятую, мы указываем имя объекта, куда будет писаться лог. Указываем тип аппендера – консоль (ConsoleAppender). Указываем, куда именно будем писать – System.out. Задаем класс, который будет управлять шаблонами записей – PatternLayout. Задаем шаблон для записи, который будет использоваться. В примере выше это дата и время.

Согласно настройкам – логи можно найти здесь