ЧЕРНОВИК

GoodWill

Каталог: техническое описание решения

Техническое описание решения

Сазонов Кирилл sazonovkirill@gmail.com
23.09.2012

Оглавление

Введение

Данный документ описывает общую архитектуру и технические особенности проекта Goodwill Catalog. Проект представляет собой каталог продукции, производимой компанией Goodwill.

С точки зрения клиента: Каталог позволяет находить нужные детали по их номерам, автомобилям, на которых они применяются, размерам деталей. Каталог позволяет просматривать информацию о деталях. С точки зрения сотрудника Компании, Каталог позволяет вести обширную баз автомобилей, марок, серий, двигателей, деталей, производителей деталей, номеров деталей и т.д.

Разработчик: Сазонов Кирилл
mail: sazonovkirill@gmail.com
skype: sazonovkirill
Разработка завершена и передана на сопровождение в ноябре 2012г.

Состав приложения

Система сборки

Проект построен и собирается на базе технологии maven.

Каталог состоит из 7 maven модулей:

CatalogParent

Родительский модуль. Определяет параметры сборки и версии всех сторонних компонентов/библиотек.

CatalogUtils

Служебный модуль, содержит системную функциональность, используемую другими модулями. Также содержит аспекты и подсистему их обработки, которая составляет важную часть архитектуры решения. Также содержит служебное приложение **CatalogUpdater**, предназначенное для обновления БД приложения (подробнее ниже).

Собирается в **jar**.

CatalogCore

Ядро приложения. Содержит доменную модель приложения и основные сервисы приложения.

Собирается в **jar**.

CatalogUI

Администраторская часть приложения, предназначена для ведения базы приложения. Представляет собой Swing Application.

Собирается в **executable jar**.

CatalogWeb

Веб-приложение, представляет собой клиентскую часть приложения. Клиентская часть позволяет только просматривать БД, ничего не изменяя в ней.

Собирается в **war**.

CatalogRunner

Служебное приложение, предназначенное для запуска т.н. «отчуждаемой копии» Каталог (подробнее об отчуждаемой копии ниже).

Собирается в **executable jar**.

CatalogAdapters

Служебное приложение, предназначенное для загрузки данных в Каталог из предыдущей версии каталога (загрузка из .csv файлов, которые предварительно были экспортированы из БД Microsoft Access).

Собирается в **jar**.

Расположение проекта

Система контроля версий

В качестве системы контроля версий используется Subversion 1.6.

Внимание!
Subversion 1.6 выбран потому, что с 1.7 замечен ряд глюков. Кроме того, Intellij IDEA поддерживает Subversion 1.6, а не 1.7.

Проект размещен в репозитории `https://goodwill-catalog.googlecode.com/svn/`
Весь проект лежит в **trunk**.

Таким образом, чтобы сделать checkout, необходимо выполнить:
`svn checkout https://goodwill-catalog.googlecode.com/svn/trunk/`

Подсистема хранения данных

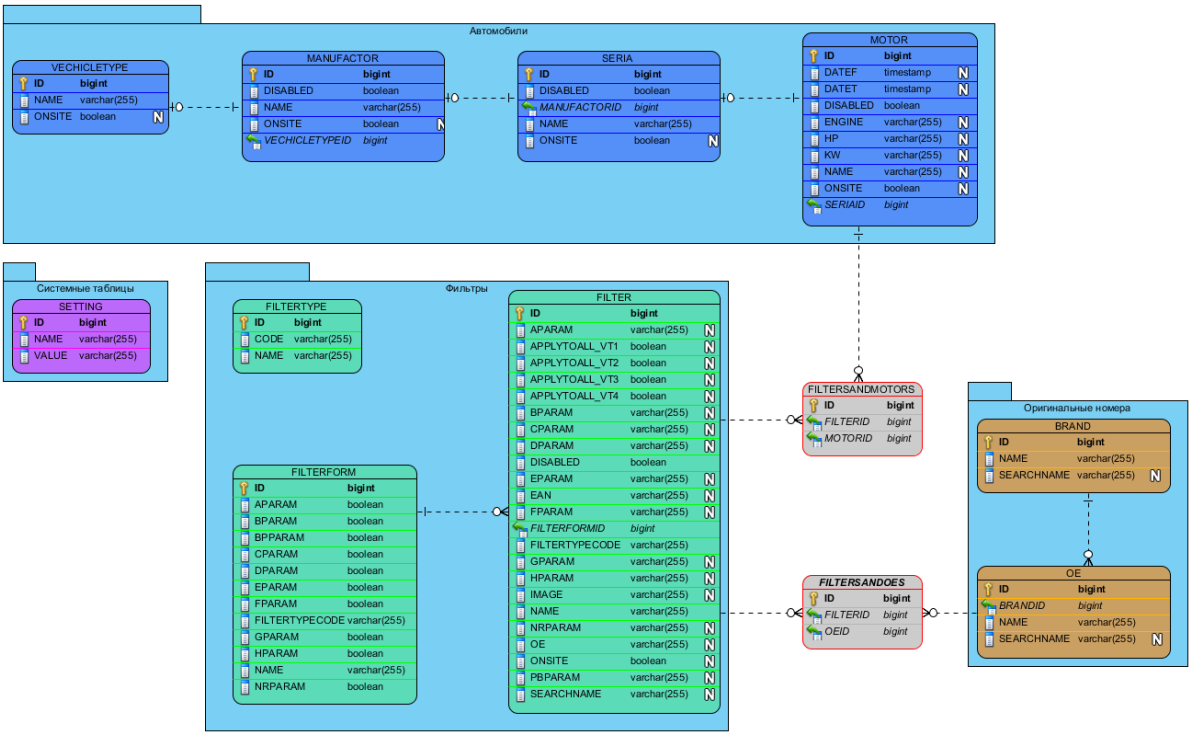
Используется СУБД H2. Она была выбрана за легковесность, простоту использования. Т.к. с базой всегда работает только один пользователь, была выбрана встраиваемая (embedded) БД.

Важно!
При работе с БД H2 важно следить, что используется самая актуальная версия JDBC драйвера. При использовании устаревших драйверов были замечены критические ошибки. На момент написания документа актуальной является версия **1.3.167**.

Критика
Сейчас есть ощущение, что стоило остановиться на какой-то более распространенной (пусть и более тяжеловесной БД), например MySQL.

Структура БД

Ниже представлена структура БД (в поставку входит файл GoodwillCatalog.vpp, который содержит структуру БД в формате Visual Paradigm Suite)



Краткое описание таблиц

Регион «**Автомобили**»

Наименование	**Назначение**
VECHICLETYPE	Типы двигателей
MANUFACTOR	Производители (марки)
SERIA	Серии (модели)
MOTOR	Моторы

Регион «**Фильтры**»

Наименование	**Назначение**
FILTERTYPE	Типы фильтров
FILTERFORM	Формы фильтров
FILTER	Фильтры

Регион «**Оригинальные номера**»

Наименование	**Назначение**
BRAND	Брэнды
OE	Оригинальные номера

Регион «**Системные таблицы**»

Наименование	**Назначение**
SETTING	Настройки приложения

Соединительные таблицы

Наименование	Назначение
FILTERSANDMOTORS	Привязки фильтров к моторам
FILTERSANDOES	Привязки фильтров к оригинальным номерам

Важно!

При рассмотрении структуры БД важно обратить внимание на денормализацию, затрагивающую таблицы FILTER_TYPE, FILTER_FORM, FILTER. С точки зрения бизнес-логики FILTER_TYPE относится к FILTER_FORM как один ко многим [1:∞] и FILTER_FORM относится к FILTER как один ко многим [1:∞]. (Другими словами для каждого фильтра существует форма фильтра, к которой относится ильтр и формы фильтров группируются по типам фильтров). Однако, таблица FILTER содержит связь не только с таблицей FILTER_FORM (поле FILTERFORMID), но и таблицей FILTER_TYPE (поле FILTER.FILTERTYPECODE). Вторая связь лишняя и должна игнорироваться приложением во избежание конфликтов. Изначально, она была оставлена для совместимости с БД предыдущего каталога.

Витрина данных

В приложении используется две витрины данных (Data Mart). Описание витрин находится в каталоге **`etc/sql`** репозитория. Более подробно см.

При подкладывании в Tomcat новой версии Отчуждаемой Копии необходимо не зыбть подложить также:

1. Актуальную БД
2. Актуальную папку IMAGES
3. Актуальный конфиг локализации core.xml

Выгрузки.

Витрины

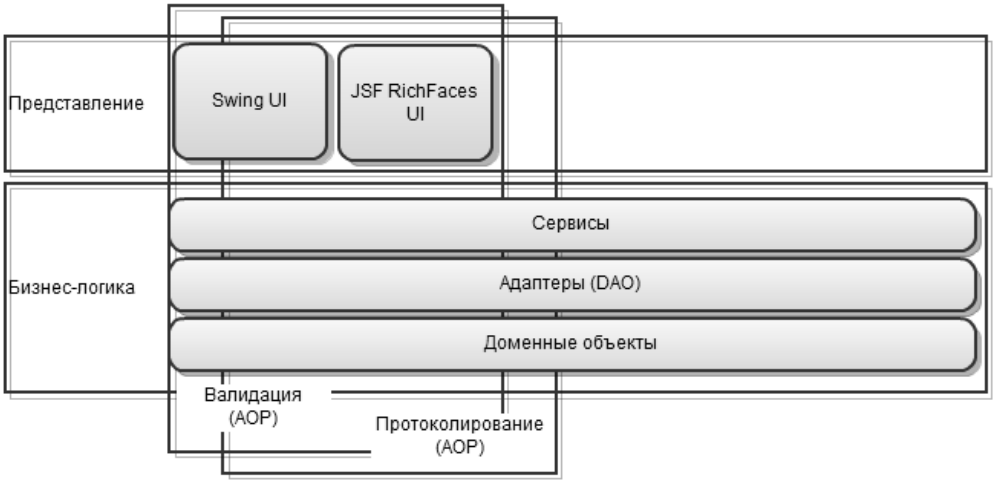
Наименование	Назначение
FullReport.sql	Полная выгрузка из Каталога
OeReport.sql	Выгрузка оригинальных номеров

Общая архитектура приложения

Технологический стек

Назначение	Технология/компонент
Платформа	Java
Язык программирования	Java
Предпочтительная IDE	Intellij IDEA
Система сборки	Maven
IoC контейнер	Guice
Представление (Presentation)	Swing (JGoodies), JSF (JSF 1.2, Facelets, Richfaces)
СУБД	H2, JDBC
Прикладные компоненты общего назначения	Apache Commons, Java Mail
Протоколирование	SLF4J, log4j
AOP	AspectJ, Spring Aspects
Кэширование	EHcache
ORM	Hibernate

Слои



Слой бизнес логики состоит из:

- Доменных объектов, каждый из которых связан с таблицей БД.
- Адаптеров, которые инкапсулируют, как правило, CRUD операции с одной сущностью/таблицей.
- Сервисов, которые реализуют бизнес-функционал приложения.

Сквозь все слои проходит валидация объектов, которая реализована через AOP посредством Spring аспектов (**`ValidationAspect`**), библиотеки HibernateValidator и аннотации **`@Managed`**.

Протоколирование также может быть реализовано посредством аннотации **`@Logged`** и AOP аналогичным образом (**`LoggingAspect`**).

Эти аннотации, аспекты и их реализации расположены в проекте **CatalogUtils** в пакетах:

- `ru.goodfil.catalog.annotations`

- `ru.goodfil.catalog.aspects`
- `ru.goodfil.catalog.validation.core`

Реализация IoC

В качестве IoC контейнера выбран Google Guice, прежде всего за легковесность. Кофигурация Google Guice называется «модулем». Есть два модуля:

Модуль для администраторской части	`ru.goodfil.catalog.ui.guice.CatalogModule`
Модуль для клиентской (веб) части	`ru.goodfil.catalog.web.utils.CatalogWebModule`

Ключевые подходы и механизмы

Кэширование

Для web части приложения включены следующие виды кэширования (реализовано на EHcache):

1. Hibernate Query Cache
2. Hibernate 2nd Level Cache

При этом следует помнить, что клиентская часть (веб часть) работает с БД только на чтение (соответственно, любое кэширование безопасно).

В будущем для оптимизации быстродействия возможно:

1. Тонко настраивать регионы для сущностей (Hibernate 2nd Level Cache).
2. Тонко настраивать регионы для запросов (Hibernate Query Cache).
3. Вручную оптимизировать запросы в коде.
4. Добавлять необходимые витрины (Data Marts).
5. Использовать ручное кэширование (manual caching), аналогичное Spring Cache Abstraction.

В-целом важно понимать, что на текущих объемах данных быстродействие можно наращивать очень эффективно. (Текущий объем БД примерно 100Мб).

Кэширование для администраторской части отключено (т.к. настройка кэширования достаточно трудоемка, а требования по оптимизации производительности отсутствуют).

Регионы для клиентской части настраиваются в файле **`ehcache.xml`**.

Аспекты

В приложении используются AOP (Aspect Oriented Programming) подходы для валидации и протоколирования.

Валидация

Концептуально, задумка была таковой:

Хочется уметь писать так:

```
void updateVechicleType(@NotNull @Valid final VechicleType vechicleType)
```

т.е. задавать с помощью аннотаций условия корректности параметров функции.

И так:

```
@Managed
public class AnalogServiceImpl implements AnalogsService {
    @NotNull
    @Inject
    private BrandAdapter brandAdapter;

    @NotNull
    @Inject
    private OeAdapter oeAdapter;

    @NotNull
    @Inject
    private FiltersAndOesAdapter filtersAndOesAdapter;

    @Logged
    @ValidateBefore
    @Override
    public List<Brand> getBrands() {
        return brandAdapter.getAll();
    }
```

т.е. задавать с помощью аннотаций условия корректности состояния (state) объекта.

Это было реализовано посредством:

1. Аннотации **@Managed**, которая активирует данный функционал для класса.
2. Аннотаций из библиотеки Hibernate Validator (**@NotNull**, **@NotEmpty** и др.)
3. Аннотации **@Valid**, которая говорит о том, что при проверке состояния объекта данное поле класса также должно быть проверено на валидность (рекурсивно).
4. Аннотаций **@ValidateAfter** и **@ValidateBefore**, говорящих о том, что объект должен быть проверен на валидность до или после выполнения аннотированного метода.

Критика
Данный подход оказался исключительно удачным, сделал код гораздо понятнее и чище, с единственной оговоркой: надо не забывать использовать данный подход повсеместно (только тогда он себя оправдывает).

Протоколирование

Протоколирование также может быть реализовано с помощью аннотаций. Для этого реализована аннотация @Logger, говорящая о том, что вызов метода должен быть запротоколирован (протоколируется имя метода и его аргументы).

На текущий момент реализация этой аннотации довольна слабая и может быть существенно расширена.

Вообще, АОР подход оказался исключительно успешным. Рекомендуется всячески развивать его.

Другие аннотации

Помимо вышеперечисленных есть ряд аннотаций, которые объявлены, но никак не реализованы (они зарезервированы на будущее). Это аннотации:

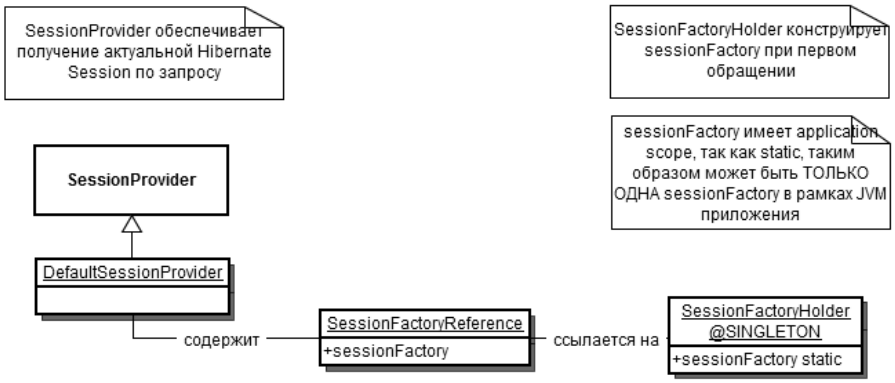
Наименование	Какое назначение предполагалось
@Init	Аннотируется метод, предназначенный для инициализации состояния объекта (замена классическим конструкторам объектов из ООП, которые не

	могут быть эффективно использованы в JavaEE в силу специфики технологии).
@Clear	Аннотируется метод, предназначенный для очищения состояния объекта (замена классическим деструкторам и финалайзерам).
@ManagedBean	Аннотируются бины JSF, предназначена для использования в качестве фасада на случай смены технологии IoC для JSF бинов (JSF Managed Bean, Spring Bean, Seam Component и т.д.)
@PageAction	Аннотируются методы JSF бинов, которые вызываются непосредственно со страниц (xhtml).

Работа с Hibernate Session

Текущая реализация работы с сессией Hibernate является максимально простой, хотя и не совсем правильной. Важно понимать, что на текущий момент в режиме чтения/записи с БД работает НЕ БОЛЕЕ чем 1 пользователь (и то, только в администраторской части). Если этот факт изменится, будем необходимо выполнить рефакторинг механизма получения сессии Hibernate.

Схема работы с Hibernate Session представлена ниже:



Такой подход имеет следующие недостатки (drawbacks):

1. Может быть только одна Hibernate SessionFactory на приложение.
2. Механизм получения актуальной сессии из SessionFactory не задан явно.

Описание дополнительных модулей

Отчуждаемая копия

Это функционал, который позволяет записать каталог на диск/флешку и далее установить его на компьютере как приложение. При этом фактически на компьютер пользователя устанавливается JRE + Tomcat и стартует Tomcat, в котором развернут Каталог. Работа с Каталогом выполняется через браузер, после окончания работы Tomcat останавливается.

Такая схема работы, как ни странно, оказалось наиболее работоспособной и подходящей.

Для создания инсталлятора отчуждаемой копии использовался продукт Advanced Installer.

Для запуска отчуждаемой копии (т.е. фактически запуска Tomcata и открытия бразуера) предназначен модуль **CatalogRunner**.

Как стартует отчуждаемая копия

Первым вариантом было написать скрипт, который запускал бы Tomcat в бэкграунде. Однако обнаружились проблемы:

1. Решение с «невидимым» окном cmd, с помощью VBScript не работало в WindowsXP.
2. Не было возможности написать какую-ту более сложную логику, чем просто запуск Tomcata на cmd. (А например, надо было, если приложение уже запущено, выдавать об этом предупреждающее сообщение, чтобы не запускать вторую копию Tomcata, которая понятно не запустится, потому что порт уже занят первой.) По-крайней мере, писать такую логику на cmd не хотелось.

Поэтому я решил запускать Tomcat прямо из своего приложения, через вызов

```
org.apache.catalina.startup.Bootstrap bootstrap = new Bootstrap();
bootstrap.setCatalinaBase("apache-tomcat-6.0.32-catalog");
bootstrap.start();
```

При этом, все либы томката я просто положил в CLASSPATH своего приложения (что может и не совсем корректно, но работает).

Сборка инсталлятора отчуждаемой копии

Для сборки инсталлятора отчуждаемой копии (файла msi), необходимо воспользоваться любым сборщиком инсталляторов, например Advanced Installer. В директории etc/aip находится проект для Advanced Installer 9.

Каталог приложения должен содержать:

.database – папка с БД H2 (необходимо проверить, что версия базы актуальна)
apache-tomcat-catalog – папка с развернутым на Tomcat'е каталогом (необходимо не забыть подложить актуальную папку IMAGES в webapps)
portable – папка с приложением CatalogRunner (его зависимости должны лежать в lib, сам **CatalogRunner** должен быть собран mavenом)
cjdk1.6.0_25 – папка с JDK
icons – папка с иконками приложения

catalog_debug.bat – отладочный файл для пуска Tomcata в консоли
cmdow.exe – приложение для запуска Tomcata **без** консольного окна

В проекте инсталлятора должно быть указано, что необходимо создать ярлык для запуска Каталога вида:
`/run /hid cjdk1.6.0_25\bin\java -Xms64m -Xmx128m -XX:PermSize=64m -XX:MaxPermSize=128m -jar portable/catalog.runner-1.1.15.0.jar CatalogRunner`

При подкладывании в Tomcat новой версии Отчуждаемой Копии необходимо не зыбть подложить также:

4. Актуальную БД
5. Актуальную папку IMAGES
6. Актуальный конфиг локализации core.xml

Выгрузки

В рамках работы над Каталогом были разработаны две выгрузки (функционал, позволяющий выгружать часть данных Каталога в файл Microsoft Excel). Из-за большого объема обрабатываемых данных выгрузки запускаются на СУБД (т.е. реализованы в виде «серверной» логики, DataMart).

<<Сюда добавить описание выгрузок>>

Ссылки

Apache Maven
http://maven.apache.org/

Visual Paradigm Suite
http://www.visual-paradigm.com/

Google Guice
http://code.google.com/p/google-guice/