Что такое ORM?

Возможно, кое-кто из читателей уже сталкивался с этим термином, однако, полагаю, таких будет немного, поскольку вряд ли прагматичный инженерный ум программиста подскажет ему читать статью по теме, которая ему и так уже хорошо знакома из практики. Хотя такое чтение, бесспорно, и будет весьма полезным, однако есть тысяча более важных дел. Впрочем, речь сейчас не о них, равно как и не о пользе чтения. О чём же тогда? Естественно, об ORM.

У каждой благозвучной аббревиатуры существует множество расшифровок. Мы будем говорить о той из них, которая звучит как object-relational mapping, или, если записать то же самое, но только уже по-русски, - объектно-реляционная проекция. Термин, что и говорить, для непосвящённого звучит ой как замысловато, однако, на самом деле, ничего концептуально сложного в нём нет.

Большая часть языков программирования, которые используются сегодня в индустрии программирования, относятся к объектно-ориентированным языкам. Это значит, что программист, создавая программы на них, оперирует с объектами - некоторыми абстрактными сущностями, имеющими некоторые свойства и позволяющими применять по отношению к себе некоторые методы. При всём при этом базы данных, которые используются для хранения информации об этих объектах, являются реляционными.

За примерами далеко ходить не надо - возьмём любой тематический интернет-магазин, написанный, скажем, на том же PHP. Все товары, содержащиеся в списке, удобно сделать объектами и приписать им определённые поля, зависящие от вида товара. Для мобильного телефона это могут быть размер экрана и ёмкость батареи, для автомобиля - пробег и год выпуска, а для лазерной мышки - время отклика сигнала. При считывании информации о товаре из такой базы программисту приходится писать много вспомогательного кода, который занимается считыванием определённых полей из базы, и присвоением считанных значений полям соответствующих объектов. Хорошо, если вы можете сделать это один раз - считав всё, успокоиться и пользоваться полученной информацией все последующие разы. А если нет, и считывать нужно неоднократно, да ещё и, не дай Бог, записывать? Вот уж тогда объём рутинного кода возрастает довольно ощутимо.

В общем-то, выходов из этой ситуации может быть несколько. Скажем, можно использовать вместо реляционной СУБД объектно-ориентированную - то есть, такую, в которой информация хранится не в виде привычных всем таблиц, а в виде точно таких же объектов, какими оперируют программисты в своём коде. Что ж, идея хороша, тем более, что существуют системы управления базами данных, позволяющие решать эту проблему именно таким способом. Однако само по себе использование объектно-ориентированной СУБД приводит к целому ряду новых проблем. Реляционные СУБД применяются так широко вовсе не потому, что они неудобны для программистов - они надёжны, быстры и, самое главное, привычны.

Может, есть какой-нибудь ещё способ решения обозначенной проблемы? Конечно, есть. Он, собственно говоря, и называется ORM. Заключается он в применении специальных фреймворков или библиотек, которые сами занимаются связыванием объектов в программе и записей в таблицах базы данных. Благодаря им программист частично избавляется от надоедливой рутины и получает больше времени для того, чтобы думать над более существенными аспектами реализации проекта.

ORM-решения имеют много преимуществ, по сравнению с "ручной" работой с базами данных в проекте. Они позволяют оптимизировать количество запросов к базе данных, но при этом избежать загрузки избыточных на данный момент для приложения данных. Кроме того, благодаря единому для всех используемых СУБД API-интерфейсу, который предоставляет ORM-фреймворк, в случае необходимости смены СУБД по желанию заказчика или просто из-за чрезмерного роста количества данных очень легко можно перейти с одной СУБД на другую, поскольку все SQL-диалекты уже реализованы в фреймворке его разработчиками, и программисту, использующему данный фреймворк, нет необходимости долгими зимними вечерами изучать каждый из них, чтобы быстро и успешно перевести приложение в случае острой необходимости с одной СУБД на другую. Хотя, конечно, единое API удобно не только в таких экстремальных случаях. Об этом будет ещё немного сказано дальше.

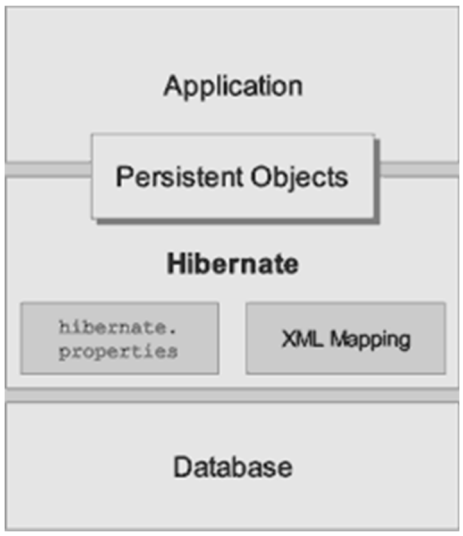
ORM и Java

Java - это такой язык программирования, на котором в наши дни пишут всё. Ну, или почти всё - начиная от тех самых пресловутых интернет-магазинов и заканчивая сложными распределёнными системами корпоративного уровня. Драйверов на нём, конечно, не пишут, но он на это, правда, и не претендует.

Современные интернет-приложения весьма и весьма активно используют базы данных - само собой, реляционные, и чаще всего MySQL. Потому использование ORM-фреймворка в таких приложениях - не роскошь, а средство спокойного существования разработчиков. К счастью, конечно же, такой фреймворк для Java существует. Называется он, как можно увидеть из названия этой статьи, Hibernate, а найти его официальный сайт можно по адресу hibernate.org. Распространяется Hibernate на условиях GNU Lesser General Public License, то есть, говоря по-русски, этот фреймворк просто свободнее свободного - его можно применять в любых приложениях (в том числе и в коммерческих), работать с его исходными текстами, и т.д. и т.п.

Стоит заметить, что, хотя Hibernate - далеко не единственный ORM-фреймворк для Java, он пользуется значительной популярностью среди программистов, поскольку это мощный, хорошо отлаженный и проверенный на множестве реальных проектов программный продукт.

Архитектура простого приложения, использующего Hibernate, тоже, простите за тавтологию, достаточно проста - вы можете увидеть её на иллюстрации к статье. Как и ожидалось, Hibernate является связующим звеном между самим приложением и данными, хранящимися в базе. При этом обратите внимание на два небольших более тёмных прямоугольника на фоне прямоугольника с надписью "Hibernate" - они символизируют конфигурационные файлы, которым мы с вами должны будем уделить самое что ни на есть пристальное внимание.



Работа с Hibernate

Как работает Hibernate? Загрузив свою собственную конфигурацию из файла hibernate.cfg.xml, он собирает информацию о классах, которые разработчик решил "замапить" в базу данных из специальных конфигурационных файлов, которые записываются в формате XML. После загрузки всех конфигурационных файлов можно создавать сессии подключения к базе данных с помощью "фабрики сессий", и работать с ними - изменять данные, добавлять, удалять. В общем, как говорится, воротить с данными всё, что только душе будет угодно.

Для начала давайте поговорим о конфигурировании Hibernate. Посмотрите на листинг - там вы можете увидеть пример конфигурационного файла hibernate.cfg.xml для Hibernate; комментарии в нём оставлены для удобства чтения. Мы с вами подробно разберём всё, что в нём написано.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<!- Database connection settings ->
<property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
<property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
<property name="connection.username">sa</property>
<property name="connection.password"></property>
<!- SQL dialect ->
<property name="dialect">org.hibernate.dialect.HSQLDialect</property>
<!- Echo all executed SQL to stdout ->
<property name="show_sql">true</property>
<mapping resource="events/Event.hbm.xml"/>
<mapping resource="events/Person.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Первые две строки, как водится, пропускаем - они стандартны для всех XML-файлов и не являются специфическими для конфигурационных файлов Hibernate. hibernate-configuration - это элемент, являющийся корневым для всего документа; session-factory описывает параметры данной конкретной "фабрики сессий", с которой, в конечном итоге,

и будет работать наше приложение. Свойства, озаглавленные в комментарии как "Database connection settings", действительно указывают настройки подключения к БД: connection.driver_class - это класс JDBC-драйвера, обеспечивающего подключение к нужной СУБД (этот драйвер тоже, как вы понимаете, должен быть включён в проект); connection.url - это адрес для подключения к базе, который передаётся драйверу; connection.username и connection.password - это, соответственно, логин и пароль того пользователя, от имени которого мы осуществляем подключение к базе данных. Свойство dialect определяет класс SQL-диалекта, который Hibernate будет использовать для подключения к базе данных. Использование свойства show_sql удобно при отладке - если его включить (выставить в true), то Hibernate будет выводить информацию обо всех посылаемых к базе данных запросах на консоль. Вне отладки это свойство лучше делать false, поскольку в реальных приложениях пользователю не то что не нужно видеть запросы, а даже и вредно с точки зрения безопасности приложения. Замыкают наш конфигурационный файл два элемента, которые начинаются с "mapping resource...". Это - пути к конфигурационным XML-файлам, в которых описаны способы представления тех или иных классов Java-приложения в реляционной базе данных, с которой это приложение через Hibernate будет работать. Эти конфигурационные файлы могут быть довольно сложны из-за иерархических отношений между описываемыми в них классами, а потому о них мы с вами подробно поговорим уже во второй части этой статьи.