Вестник Сыктывкарского университета.
Сер.1. Вып.8. 2008

УДК 519.68

SOA СИСТЕМА ХРАНЕНИЯ И ПОИСКА ИНФОРМАЦИИ (ПЛАТФОРМА MAGNET)

Е.А. Ключников

Рассматриваются преимущества Java реализации SOA системы хранения и поиска данных перед классическими SQL серверами на примере платформы MAGNET.

1. Введение

Бурное развитие в области микропроцессорных разработок непосредственно влечёт за собой повсеместное их распространение. На данный момент уже невозможно представить себе функционирование промышленных предприятий, транспортных предприятий, банков, государственных учреждений без использования компьютерной техники. Однако, микропроцессорная техника, сама по себе, не решает задач ввода, обработки и распределения информации, координирования информационных потоков. Данные задачи решаются специализированным программным обеспечением (ПО) — информационными системами.

Создание универсальной информационной системы невозможно, поскольку в каждой области человеческой деятельности существует множество нюансов, порой конфликтующих между собой. Поэтому для каждой сферы деятельности создаются информационные системы ориентированные на определённый круг задач, и предоставляющие пользователю максимально удобное «рабочее место».

Создание информационных систем — трудоемкое и многостадийное мероприятие. Однако анализ существующих систем позволяет выделить фрагменты, которые можно использовать повторно, выявить подсистемы, хорошая реализация которых может в дальнейшем сильно сократить трудозатраты, необходимые для создания новых и поддержания существующих информационных систем.

Применение SOA[1] позволяет сделать процесс создания информационных систем более эффективным. В самом широком смысле, SOA — это архитектура, обеспечивающая интеграцию и связность. SOA развивается, превращая Интернет в вездесущую утилиту, доступную для всех способов обмена данными, коммерции и коммуникаций. Это архитектура, предназначенная, по самой своей природе, для использоавния естественно возникающей гетерогенности систем и инфраструктур. Она развивается не только для того, чтобы помогать системам связываться друг с другом, но и для обеспечения возможности взаимодействия во множестве измерений, в бесконечно разрастающемся массиве требований заказчиков, от простого использования web-коммерции до более сложных запросов промышленных вертикалей.

В результате обобщения опыта создания информационных систем компанией ООО «Магнетософт» была создана платформа MAGNET.

MAGNET — это модульная сервис-ориентированная платформа с открытым исходным текстом для создания информационных систем. Модули платформы написаны на языке Java 2 Enterprise Edition.

Основные сервисы, предоставляемые данной платформой:

- надёжное хранение информации;
- быстрый и удобный доступ к хранимой информации;
- распространение сообщений и оповещений между модулями;
- выполнение «технологических процессов»[2];
- «сквозная аутентификация[3]»[4];
- авторизация[5] на основании динамических условий.

[1]SOA — Service-Oriented Architecture, см. http://en.wikipedia.org/wiki/Service-oriented_architecture

[2]Технологический процесс (workflow) — набор действий, выполняемых пользователем и выполняемых автоматически, связанных по определённой схеме; часто используется для организации документооборота.

[3]Аутентификация — процесс проверки того, что пользователь действительно является тем, кем представляется.

[4]Сквозная аутентификация (single sign-on) — особая методика аутентификации, при использовании которой результаты аутентификации в одной системе доступны всем системам той же группы.

[5]Авторизация — процесс определения прав пользователя на различные действия над объектами.

При создании масштабируемой информационной системы ключевым моментом является способ хранения и извлечения информации. В данной статье рассмотрены модели хранения данных платформы MAGNET, и их преимущества при создании информационных систем.

2. Недостатки реляционных СУБД и их исторические предпосылки

Наиболее распространённым и общепринятым способом работы с информацией, на момент написания статьи, является использование реляционных систем управления базами данных (СУБД, DB). При таком подходе данные хранятся в виде набора таблиц, состоящих из строк с заранее определёнными полями. Каждую такую таблицу можно представить как подмножество прямого декартова произведения множеств возможных значений столбцов (доменов). Подобные подмножества часто определяют отношения (relation), откуда и идёт название — «реляционные».

Со времени создания первых реляционных СУБД и до недавнего времени сохранялась следующая ситуация: объём хранимых данных значительно превышает объём оперативной памяти. Выборка данных сопровождалась обращением ко вторичной памяти[6], скорость доступа к которой, по сравнению со скоростью доступа к оперативной памяти, чрезвычайно мала[7]. Для уменьшения количества обращений ко вторичной памяти используются специальные структуры постраничного хранения данных — Б-деревья[8].

На данный момент объём оперативной памяти вычислительных серверов позволяет размещать все хранимые данные предприятий среднего и даже крупного бизнеса. Однако системных решений, позволяющих максимально использовать данное преимущество очень мало.

Общепринятой методологией проектирования баз данных является приведение структур данных к так называемым нормальным формам (см. рис. 2а, 2б). Целью указанных преобразований является сведение к минимуму количества избыточных данных. С другой стороны, уменьшение избыточности приводит к тому, что структуры становятся менее понятными для человека, сокращаются возможности создания сложных индексов, используемых для оптимизации отбора данных.

---

[6]Вторичная память (secondary storage) — устройства долговременного хранения данных; наиболее распространённое устройство — жёсткий диск (hard disk drive).

[7]Типичное время доступа (access time) к жесткому диску составляет 10 мс.

[8]Б-дерево (B-tree) — корневое сбалансированное дерево с задаваемой степенью ветвления; Б-дерево со степени 1001 и высоты 2 позволяет за 2 обращения ко вторичной памяти осуществить поиск среди более чем миллирда ключей.

Запросы к СУБД осуществляются посредством специализированного языка запросов (SQL[9]). Существует множество реализаций СУБД, воспринимающих различные диалекты SQL. Соответственно перевод готового программного решения с одной SQL на другую осложнён изменением заложенных в решение запросов, согласованием типов хранимых данных; а также необходимо осуществить корректный перенос данных из одной системы в другую. Такой перевод может потребоваться, если окажется, что текущая используемая СУБД не обладает достаточной производительностью или не удовлетворяет заказчика своей стоимостью (подавляющее большинство производительных СУБД являются коммерческими продуктами).

Для связи с СУБД используются «закрытые» протоколы, то есть протоколы, о которых нет информации в общем доступе (см. рис. 1а); таким образом, для того чтобы связаться с СУБД, необходимо использовать исполнимый код, представленный разработчиками СУБД. Данный фактор приводит к тому, что разрабатываемое решение будет платформо-зависимым (чаще всего привязка идёт к коммерческой ОС Microsoft Windows Server и языкам программирования, реализованным на данной платформе).

Если резюмировать всё вышесказанное, то можно сказать, что реляционные СУБД ориентированы на предоставление низкоуровневых сервисов хранения и выборки данных в условиях ограниченных ресурсов. Системы управления данными, рационально использующие ресурсы современной микропроцессорной техники, могли бы предоставлять более высокоуровневые и удобные, а также более производительные сервисы.

3. Система объектного хранения данных

Платформа MAGNET предоставляет несколько сервисов для хранения данных. Сервисы развивались в соответствии с пожеланиями программистов к удобству использования и требованиями заказчиков к производительности и масштабируемости.

Модули платформы являются web-сервисами (WS[10]). Web-сервис — это программа, вызов методов которого происходит посредством передачи сообщений открытого протокола SOAP[11]. Передача сообщений возможна через различные открытые сетевые протоколы, например HTTP[12]. Таким образом использование сервисов платформы возможно

[9]SQL — Structured Query Language

[10]WS — Web Service, см. http://www.w3.org/2002/ws

[11]SOAP — Simple Object Access Protocol, см. http://www.w3.org/TR/soap

[12]HTTP — HyperText Transfer Protocol, см. http://tools.ietf.org/html/rfc2616

из большинства современных языков программирования, независимо от среды их исполнения. Более того, обращение к сервисам возможно через сеть интернет, поскольку использование протокола HTTP позволяет избежать проблем связи между клиентом и сервером (см. рис. 1б).

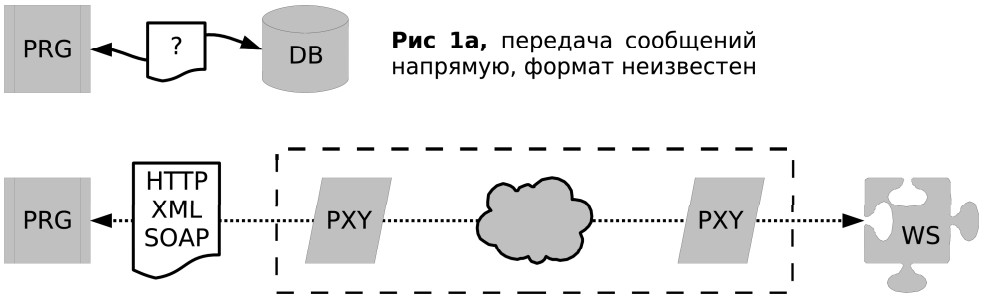

Рис 1а, передача сообщений напрямую, формат неизвестен

Рис 1б, передача сообщений через веб-сервисы; распространение может проходить в сетях без прямого разрешения конечных точек благодаря протоколу HTTP; структура данных задаётся при помощи XML, а информация о вызове – протоколом SOAP.

***PRG** – пользовательская программа; **DB** – сетевая СУБД;*
***WS** – веб-сервис; **PXY** – прокси-сервер*

Первым пожеланием программистов было использование объектов языка программирования вместо непосредственной работы с таблицами (см. рис. 2в). Модуль, реализующий данное пожелание, также ограждал бы готовые решения от деталей реализаций хранилища данных; то есть в случае изменения принципов хранения все необходимые исправления будут внесены только в модуль хранения.

Использование объектов позволяет скрыть от программиста использование транзакций. Действительно, в подавляющем числе случаев транзакции используются для согласованности данных, относящихся к одной сущности; в случае использования объектов согласованность данных поддерживается модулем хранения.

Объект, с одной стороны, представляет собой не избыточную структуру, при представлении его в табличном виде, а с другой, удобный способ управления данными.

Первая реализация модуля хранения (Entity Manager, далее EM) использовала технологии JAXB[13] для преобразования объектов в XML[14]

[13]JAXB — Java Architecture for XML Binding, см. https://jaxb.dev.java.net
[14]XML — eXtensible Markup Language, см. http://w3.org/XML

(данные, пригодные для передачи по сети) и Hibernate[15] для хранения в реляционной СУБД. Для того, чтобы начать использовать уже развёрнутый модуль хранения, достаточно было скопировать пакет с классами используемых объектов в рабочую папку модуля. На объекты хранения накладывалось всего три ограничения:

- хранимые поля должны удовлетворять спецификации Java Bean, то есть должны быть доступны через методы «get» и «set»;
- объект должен обладать хранимым строковым полем «id» (идентификатор), используемым и назначаемым модулем;
- поле должно быть примитивом (строка, число, время,...), либо списком (List) примитивов, либо другим объектом хранения, либо списком других объектов хранения.

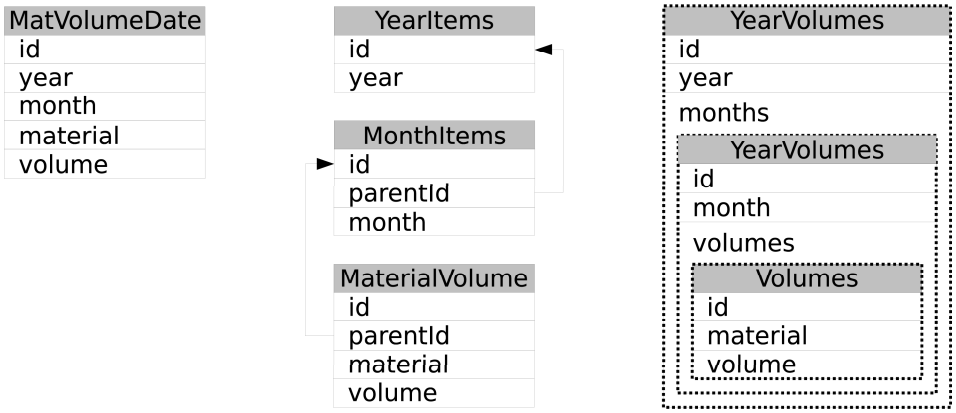

Рис 2а, «плоская» модель данных

Рис 2б, «нормализованная» модель данных

Рис 2в, «объектная» модель данных

Благодаря такому подходу удалось сократить затраты времени на создание объектной модели приложения и поддержание методов доступа к данным в актуальном состоянии. Однако недостатком модуля являлось то, что он не позволял производить выборки данных, используя язык запросов. Для решения данного недостатка был создан модуль Search Query (далее SQ).

Модуль SQ получал данные из модуля EM, преобразовывал их к табличному виду и сохранял в специальной базе данных. СУБД

[15]Hibernate — object relational model, см. http://www.hibernate.org

«HSQLDB[16]» позволяет хранить все данные в памяти сервера и производить быстрые выборки. Выключение сервера приводит к потере данных, сохранённых в СУБД. Запись и обновление информации соответствовали схеме на рис. 3а.

Выборка данных осуществляется в два этапа: сначала производится выборка списка идентификаторов объектов при помощи модуля SQ, затем получение объектов из модуля EM по списку идентификаторов (см. рис. 3б). Двухстадийная выборка данных позволяет создавать приложения с быстрым откликом на запрос большого количества данных. Пользователю системы нет необходимости получать все данные выборки сразу, поскольку он не может воспринять большие объёмы информации сразу; при этом, время до получения первой порции данных должно быть минимальным. При получении списка идентификаторов система может запросить объекты по некоторому подмножеству списка идентификаторов, а оставшуюся часть списка сохранить в памяти для последующего использования (при постраничном выводе информации или динамической подгрузке данных).

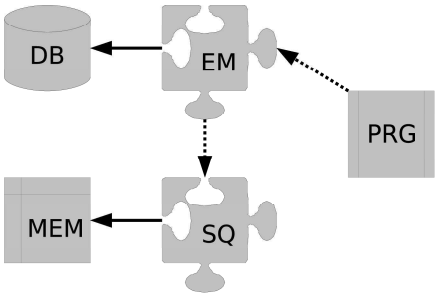

Рис. 3а, сохранение данных

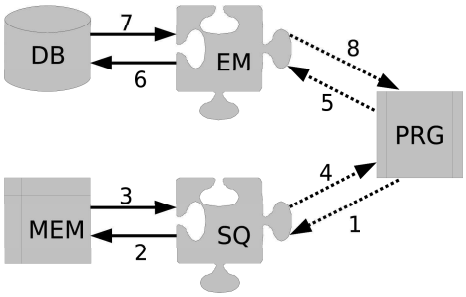

Рис. 3б, получение данных

Пунктирной линией обозначены вызовы процедур (передача данных) через web-сервисы. ***MEM*** *– хранилище данных в памяти*

Указанная ранее модель хранения данных обладает следующими недостатками:

- отсутствие масштабируемости (возможности повышения производительности при возрастании нагрузки на систему);
- долгий отклик при сохранении данных большого объёма;

[16]HSQLDB — Hypersonic SQL DataBase, см. http://hsqldb.org

- несогласованные выборки (только что сохранённый объект может отсутствовать в выборке).

Для решения указанных проблем было решено изменить модель хранения и использовать иные технологии (для увеличения производительности).

В новой модели каждый хранимый объект имеет метку версии. При запросе на сохранение данных вызывающей стороне возвращается номер версии, в которой данные будут иметь актуальное значение. Сам запрос помещается в очередь, распространяется между несколькими модулями хранения и поиска и выполняется асинхронно. Таким образом решаются сразу несколько проблем.

Во-первых, запрос на выборку поручается наименее занятому модулю поиска. В дальнейшем запрос на получение объектов поручается наименее занятому модулю хранения. Более того, серверы поддерживающие работоспособность модулей могут быть географически удалены; данные будут поступать из ближайшего источника, что, в некоторых случаях, многократно уменьшает время отклика. То есть выполняется требование масштабируемости.

Во-вторых, при неработоспособности одного или нескольких модулей хранения или поиска данные остаются незатронутыми. После восстановления работоспособности данные восстанавливаются из исправных хранилищ. Это увеличивает надёжность хранения.

Версионность хранения позволяет отследить изменения и решает проблему несогласованных выборок. Асинхронное выполнение запросов на изменение данных приводит к тому, что отклик системы становится практически мгновенным.

Для хранения данных в модуле хранения, взамен технологии Hibernate, сохраняющей данные в реляционной СУБД, использована специализированная система хранения XML данных — BerkleyDB[17]. Это позволило многократно сократить время необходимое на сохранение объектов и воссоздание объектов из хранимых данных. Дело в том, что объекты фактически представляют собой иерархическую типизованную структуру. XML идеально подходит не только для передачи, но и для хранения такого рода данных.

Для эффективного извлечения данных из объектов, преобразования их в строки XML и произведения обратных действий используются особенности языка Java, выгодно отличающие его от других языков

[17]BerkleyDB — см. http://www.oracle.com/technology/products/berkeley-db

программирования — интроспекция, динамическая загрузка классов и динамическая компиляция.

Интроспекция — это технология, позволяющая программным путём изучать строение классов[18] и вызывать их методы. Используя интроспекцию можно получать данные о строении объектов хранения во время исполнения. Таким образом устраняется необходимость предварительного конфигурирования системы хранения. Ручное конфигурирование, зачастую, приводит к появлению критических ошибок, проявляющихся на поздней стадии разработки информационной системы.

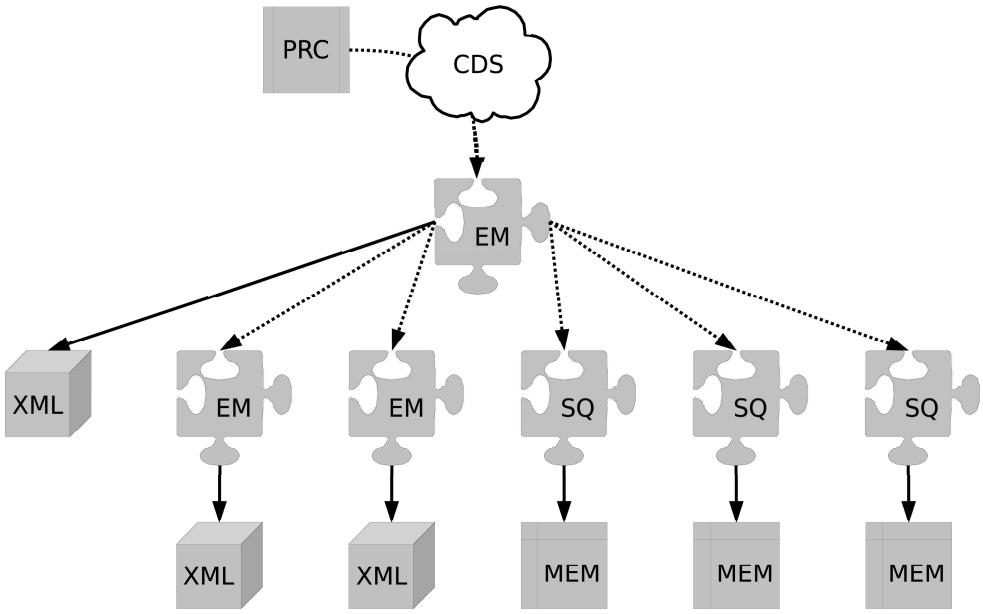

Рис. 4, распространение запроса на сохранение данных

CDS *– Catalogue & Distribution Service, служба обнаружения модулей и распределения запросов*

Динамическая загрузка классов позволяет управлять процессом определения используемых исполнимых классов. Исполнимые классы, в таком случае, могут быть получены из внешних источников или даже сгенерированы «на лету». Использование данной технологии даёт возможность выполнять высокопроизводительные процедуры, определяе-

[18]Класс — основная структурная единица языка Java; класс содержит описания своих полей (которые имеют тип класса или примитива) и методов; все объекты являются экземплярами классов.

мые входными данными, вместо выполнения низкопроизводительных универсальных процедур.

Динамическая компиляция — одна из наиболее значимых особенностей языка Java. Язык Java является интерпретируемым языком. Однако во время исполнения программы виртуальная машина Java (интерпретатор) накапливает данные о производительности исполняемого кода. Когда накопленные данные указывают на наличие «узкого места» (например часто используемого фрагмента кода), виртуальная машина производит трансляцию фрагмента в машинный код, выполняющийся с максимальным быстродействием на данном микропроцессоре. При трансляции кода используются методы оптимизации, недоступные трансляторам обычных языков программирования, по причине чрезвычайного разрастания итогового исполнимого кода при применении оптимизации целиком ко всей программе. Это происходит потому, что у компиляторов обычных языков программирования нет данных о реальном использовании тех или иных участков программы.

Использование указанных особенностей языка позволяет создавать производительные программы, работающие с данными заранее неопределённой структуры, или программы, эффективно работающие в изменяющихся условиях.

Применение СУБД «H2[19]» вместо «HSQLDB» позволяет хранить данные и в памяти, и на постоянном носителе информации, соответственно, в случае согласованности данных исчезает необходимость полного индексирования данных. Версионность хранимых объектов позволяет производить инкрементальное обновление данных модулей SQ и EM в случае их временной неработоспособности.

Запись и обновление информации в новой системе соответствует схеме на рис. 4.

Основной задачей, решение которой позволило перейти к новой модели данных, является определение ведущего модуля EM. Наличие в инфраструктуре такого «особого» элемента автоматически решает задачу назначения версии и идентификаторов новым хранимым данным.

Каждый работающий модуль EM имеет уникальный коэффициент приоритета. Ведущий сервер определяется следующим образом (см. рис. 5а, 5б, 5в, 5г):

1. при запуске модуля, используя сервер UDDI[20], определяются все

[19]H2 — «Hypersonic» 2 Database Engine, см. http://www.h2database.com

[20]UDDI — Universal Description Discovery and Integration, инструмент для расположения описаний веб-сервисов (WSDL) для последующего их поиска; см. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

модули EM, входящие в одну логическую группу;

2. происходит попытка соединения с каждым из найденных модулей;
3. каждому модулю отправляется сообщение, содержащее коэффициент приоритета текущего модуля; ответ содержит коэффициент приоритета опрашиваемого модуля или отказ в случае, если в это время происходит процедура переопределения текущего ведущего модуля; в случае получения отказа процедура повторяется позже;
4. обновляются данные до актуального состояния (из работающих модулей);
5. после опроса может возникнуть ситуация, когда текущий ведущий модуль уже определён верно; в данном случае процедура прерывается;
6. если текущий модуль должен стать ведущим, то процедура установки ведущего модуля переходит в завершающую стадию; все активные модули уже знают о том, какой модуль должен стать ведущим, соответственно, осталось лишь отправить подтверждение об изменении ведущего модуля.

Такая процедура установления ведущего модуля таит в себе опасность неверного обновления данных. Рассмотрим ситуацию в системе с двумя модулями хранения (см. рис. 5д):

1. происходит остановка первого модуля;
2. второму модулю поступают запросы на обновление данных;
3. второй модуль модуль останавливается;
4. первый модуль стартует;
5. первому модулю поступают запросы на обновление данных;
6. второй модуль стартует.

Такой сценарий приводит к тому, что оба модуля обладают новыми данными, возможно взаимнопротиворечивыми. Подобная ситуация могла сложиться и в результате обрыва соединения между модулями. Для разрешения подобных ситуаций используются версионные данные, коэффициенты приоритета и идентификаторы объектов.

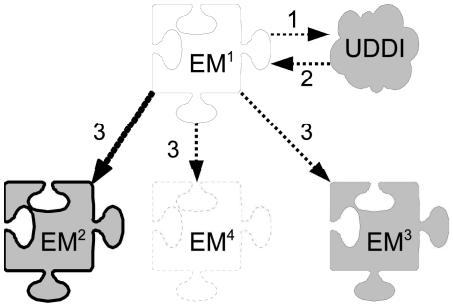

Рис 5а, обнаружение модулей и попытка подключения

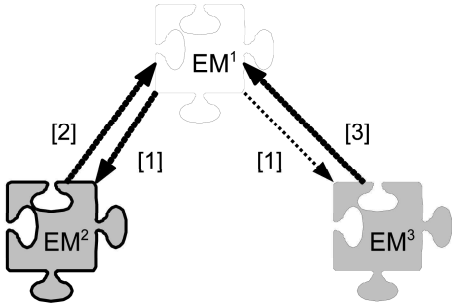

Рис 5б, обмен информацией о коэффициенте приоритета

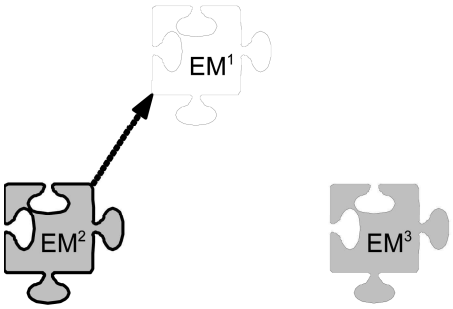

Рис 5в, обновление данных из ведущего модуля

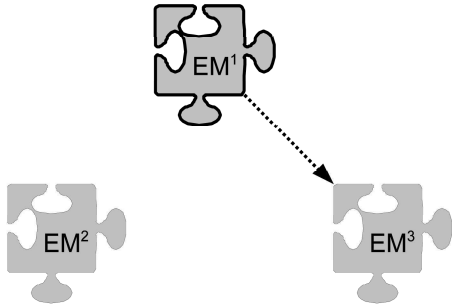

Рис 5г, подтверждение изменения ведущего модуля

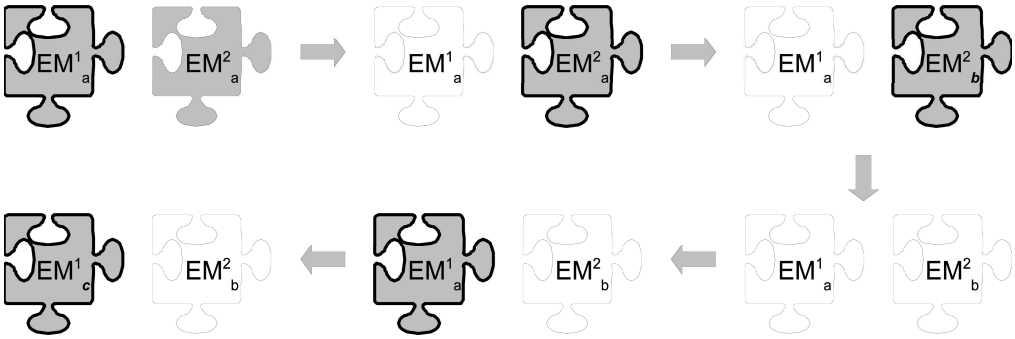

Рис 5д, последовательность изменения состояний модулей и обновления данных, приводящая к рассогласованию хранимых данных

Изменения, произошедшие со времени отключения модуля до времени старта, можно вычислить по номеру версии на момент отключения и номеру текущей версии. Изменения данных модуля с меньшим ко-

эффициентом приоритета объединяется в отдельный пакет. Пакет пересылается ведущему модулю; после применения изменений модуль с меньшим коэффициентом приоритета получает пакет суммарных обновлений, применение которого приводит к тому, что данные модулей становятся синхронными.

Если вероятность выхода из строя сервера, на котором выполняется модуль, в каждый отдельно взятый день принять равной k, а количество дней, в среднем требующееся на восстановление работоспособности, равным l, то вероятность отказа системы, состоящей из n модулей выполняющихся на различных серверах будет равна $k^n \cdot l^{n-1}$. При типичных значениях $k = 0.005$, $l = 4$, при использовании двух модулей вероятность отказа системы равна 10^{-4}, а при использовании трёх модулей — $2 \cdot 10^{-6}$.

Таким образом, использование сервисов объектного хранения платформы MAGNET можно считать удобной и надёжной альтернативой использования реляционных СУБД для хранения и обработки данных.

4. Система ассоциативного хранения

Анализ показал, что большинство информационных систем создано для решения задач классификации и упорядочения разнородных данных. Данная задача решается ассоциацией исходных данных и некоторых атрибутов, характеризующих эти данные. Особенность проектирования таких систем заключается в том, что, чаще всего, нет возможности на этапе разработки предусмотреть и заложить все возможные атрибуты для данных всех типов. В информационную систему закладывается возможность конфигурирования типов данных и их атрибутов по необходимости. Такой подход приводит к тому, что объект описания данных представляет из себя список пар «тип атрибута — значение атрибута». При использовании реляционных СУБД возникает структура хранения, схожая со структурой на рис. 6.

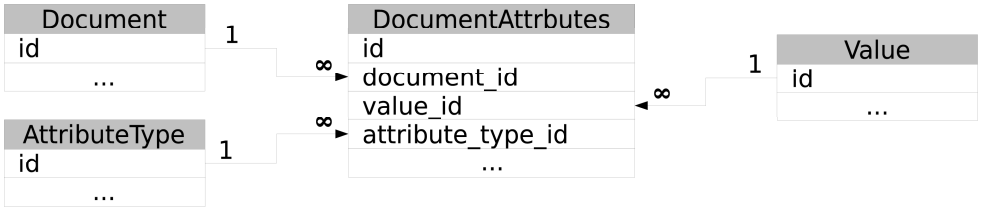

Рис. 6, структура хранения ассоциативных данных

Недостатки такой структуры хранения проявляются при реализации выборки запрошенных пользователем данных. Пользователь может накладывать условия ограничения на произвольные атрибуты и требовать данные, отсортированные по указанным атрибутам. Язык запросов реляционной СУБД не содержит операций, позволяющих выполнять отбор и сортировку на основании набора подчинённых данных как единого целого (операции применяются к отдельным элементам набора). Как следствие, для определения списка элементов, подлежащих выбору, придется либо просматривать все объекты, что невозможно, поскольку количество объектов в информационной системе, как правило, очень велико, либо выполнить серию выборок и объединить их данные. После определения списка элементов производится из извлечение из базы данных и сортировка.

Как уже было сказано ранее, подобные трудности будут возникать при создании большинства информационных систем. Логичным решением таких проблем является разработка специализированной системы хранения и обработки подобного рода данных.

Механизмы мышления человека, отвечающие за классификацию объектов и выборку связных данных называются ассоциативное мышление и ассоциативная память соответственно.

Ассоциация — это взаимная связь нескольких объектов, понятий, значений. Ассоциативный поиск — это отбор ассоциаций, связанных с объектом поиска.

Для решения задачи работы с ассоциативными данными в рамках платформы MAGNET разрабатывается модуль ассоциативного хранения (Associative Storage, далее AS).

Модуль AS кроме основной задачи решает также задачу структурирования информации. Дело в том, что наиболее распространенной структурой хранения и организации данных является дерево подчинённости (иерархия). Реляционные СУБД не предоставляют сервисов для хранения структурной составляющей данных. Модуль EM позволяет хранить объекты, находящиеся в подчинении, но, как и в реляционных СУБД, выборка ассоциативных данных является нетривиальной и ресурсоемкой задачей.

Данные, хранимые в модулем AS на постоянном носителе, имеют структуру, схожую со структурой на рис. 6. Но все данные также дублируются в памяти; при этом структура данных в памяти уже иная, позволяющая осуществлять быстрые выборки и навигацию в дереве подчинённости. Реализация механизмов работы с памятью языка Java даёт возможность создавать эффективные системы хранения оператив-

ных данных, исходный код которых не загромождён техническими деталями (что чаще всего приводит к появлению трудноотслеживаемых и трудноисправимых ошибок в обычных языках программирования).

Одна из ключевых концепций языка Java заключается в том, что у программиста нет прямого доступа к памяти компьютера, доступ к данным производится через объекты. Как следствие — отсутствует необходимость в «арифметике указателей»[21]. Обращение к самим объектам производится через указатели. Если на объект не существует указателей, то объект недоступен для использования, следовательно может быть безболезненно удалён из памяти.

Среда исполнения языка Java автоматически следит за ссылками на объекты и своевременно удаляет неиспользуемые объекты — производит сборку мусора (garbage collection). Оставшиеся объекты могут быть перераспределены в памяти таким образом, чтобы между ними не было промежутков, что приведет к выделению дополнительного непрерывного свободного участка памяти (дефрагментация памяти). С точки зрения программиста, процессы сборки мусора и дефрагментации памяти являются прозрачными, то есть никак не отражаются на исполнении программы. Если подвести итог, то наиболее сложные и опасные подсистемы приложения вынесены за рамки исходного кода, что позволяет сосредоточиться непосредственно на задачах, составляющих суть системы.

Как уже было сказано, модуль AS хранит данные в оперативной памяти, что позволяет не только делать структуры более сложными, чем табличные, но и осуществлять очень быстрые выборки и обработку всего массива данных. Благодаря этому модуль представляет сервисы полнотекстового поиска и отбора информации по сложным критериям, представляющим из себя логическое выражение, состоящее из «жестких» и «нечётких» условий. Жёсткие условия (условия равенства) используются модулем для составления начального списка ассоциаций; каждый из элементов списка проходит окончательную проверку и, в случае удовлетворения критерию, переходит в результирующий список. Результирующий список сортируется по указанным в запросе полям и конвертируется в XML структуру для упрощения обработки результатов на стороне потребителя услуг модуля.

Анализ первой реализации модуля показал, что понятие ассоциатив-

[21] Арифметика указателей — вычисление адреса данных в памяти компьютера на основе базового адреса и смещения; большинство случаев неработоспособности и уязвимости приложений является прямым или косвенным следствием использования арифметики указателей; частный случай — «переполнение буфера».

ных данных может быть расширенно, а структура хранения, напротив, упрощена. Это достигается, если принять, что значения полей ассоциации сами являются ассоциациями. В случае, если ассоциация является «примитивом», то она просто не содержит полей. Такой подход упрощает выборку данных, поскольку вместо работы с «сырыми» данными, на некоторых уровнях, можно работать с ассоциациями. Логические структуры новой реализации изображены на рис. 7а, 7б.

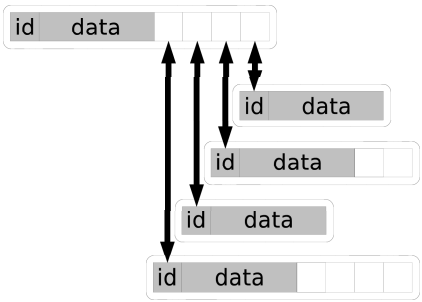

Рис 7а, свойства ассоциации являются ассоциациями

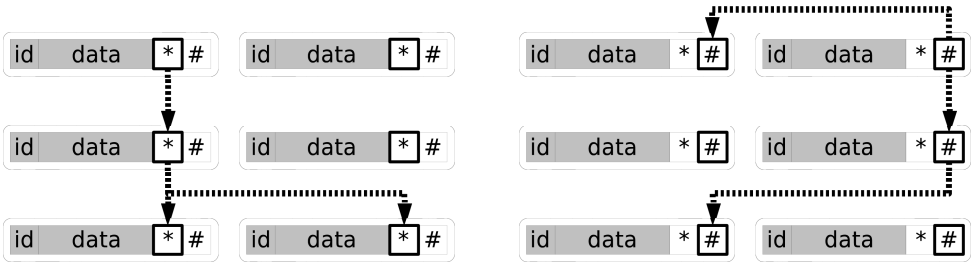

Рис 7б, ассоциации могут участвовать одновременно в нескольких иерархических структурах

Использование модуля ассоциативного хранилища позволило в кратчайшие сроки рарзработать и внедрить гибкую систему хранения структурированной справочной информации, систему классификации документов и систему хранения организационной информации.

5. Заключение

Использование сервис-ориентированной архитектуры для создания информационных систем позволяет избежать проблем интеграции, масштабируемости и поддержки возникающих перед информационными системами классической архитектуры. SOA-сервисы хранения и поиска данных, по сравнению с СУБД, имеют множество преимуществ, таких

как, динамическое описание хранимой информации, удалённый доступ, хранение структурных данных.

Применение особенностей технологии Java сделало возможным создание основных компонентов платформы MAGNET командой из 2-х человек всего за 6 месяцев. Изучение опыта эксплуатации сервисов хранения и поиска данных, входящих в состав платформы MAGNET, показало их крайнюю эффективность и пользу при разработке приложений уровня предприятия. Платформа MAGNET завоевала третье место на международной выставке Softool в номинации «Поддержка IT-структуры предприятия».

Литература

1. **Биберштейн Н., Боуз С., Джонс К., Фиаммант М., Ша Р.** Компас в мире сервис-ориентированной архитектуры (SOA): ценность для бизнеса, планирования и план развития предприятия. Пер. с англ. М.: КУДИЦ-ПРЕСС. 2007. 256 с.

2. **Кормен Т., Лейзерсон Ч., Ривест Р.** Алгоритмы: построение и анализ. Пер. с англ. под ред. А. Шеня. М.: МЦНМО: БИНОМ. Лаборатория знаний, 2004. 2-е изд., стереотип. 960 с.: 263 ил.

3. **Вебер Д.** Технология Java в подлиннике. Пер. с англ. СПб.: БХВ-Петербург, 2001. 1104 с., ил.

4. **Хабибуллин И.** Самоучитель XML. СПб.: БХВ-Петербург, 2003. 336 с., ил.

5. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб: Питер. 2001. 368 с. ил.

6. **Bruce Eckel.** Thinking in Java - 4th ed. 1408 стр. издатель Prentice Hall PTR. 2006.

7. **Sandy Carter.** The New Language of Business: SOA & Web 2.0 - 1st ed IBM Press. 320 p. 2007.

Summary

Kluchnikov E.A. SOA data store and search system

The author consider SOA data store and search system written in Java. SOA approach makes such systems scalable and easy to integrate as basis of enterprise infrastructure. Java technology makes it easier to use the full power of modern servers.

Сыктывкарский университет *Поступила 29.02.2008*