ВСТУП

Сьогодні, в епоху сучасних інформаційних технологій, більшість існуючих по всьому світу підприємств та організацій використовують комп’ютеризовані системи для різноманітних повсякденних задач, таких, наприклад, як управління складськими системами, пошук та обробка даних в базах знань, управління електронною звітністю підприємства або організації, продаж товарів через інтернет-додатки, створення Business-to-Business систем і т.д.

Не важко зрозуміти що одними з найголовніших характеристик більшості комп’ютеризованих систем є швидкість доступу до необхідної для кінцевого користувача інформації за найкоротший проміжок часу за певним критерієм та зменшення навантаження на системи, з котрими дана комп’ютеризована система взаємодіє. Слід зазначити, що кінцевим користувачем може бути як людина, так й інший інформаційний додаток, що використовує результати роботи даної системи.

Розглядаючи більшість існуючих комп’ютеризованих систем, реалізованих на мові програмування Java, можна помітити, що багато з них для роботи з інформацією використовують широко розповсюджені SQL бази даних, для взаємодії з цими базами даних використовують стандартну реалізацію SQL інтерфейсу й мають велику кількість запитів до малозмінних даних(каталоги продукції, шаблони документів, опис товарів, конфігурації систем, таблиці доступу і багато інших), що призводить до нераціонального використання інтернет трафіку (у випадку коли БД знаходиться на іншому сервері), зменшення швидкості доступу до необхідної інформації та збільшення навантаження на сервери баз даних.

Кешування малозмінних даних високонавантаженої системи допомогло б вирішити дану проблему, але, стандартна реалізацію SQL інтерфейсу мови програмування Java не має такої функціональності, в зв’язку з цим постає задача розробити та реалізувати в рамках даної мови програмування кеш, що дозволив би зменшити навантаження на SQL базу даних інформаційної системи та збільшити швидкість доступу до необхідної інформації.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

* 1. Загальні відомості
     1. Кеш

Кеш — проміжний буфер з швидким доступом, що містить інформацію, яка може бути необхідна з найбільшою ймовірністю. Доступ до даних в кеші здійснюється швидше, ніж вибірка вихідних даних з більш повільної пам'яті або віддаленого джерела, однак її обсяг істотно обмежений у порівнянні зі сховищем вихідних даних.

Використання слова «кеш» у контексті комп'ютерів походить від 1967 року, коли готувалась стаття для публікації в журналі 'IBM Systems Journal'. Стаття стосувалася захоплюючого вдосконалення пам'яті в Моделі 85, наступної в лінії IBM System/360. Редактор журналу, Lyle R. Johnson, попрохав ужити більш описовий термін, ніж «швидкохідний буфер», але іншого слова не було знайдено, тоді він запропонував «кеш». Статтю було опубліковано на початку 1968 року, автори були нагороджені IBM, їх робота широко схвалювалась і згодом була поліпшена, та назва «кеш» швидко стала стандартно вживаною в комп'ютерній літературі.

* + 1. Функціонування кешу

Кешування застосовується ЦПУ, жорсткими дисками, браузерами, веб-серверами, службами DNS і WINS.

Кеш складається з набору записів. Кожен запис асоційовано з елементом даних або блоком даних (невеликої частини даних), яка є копією елемента даних в основній пам'яті. Кожен запис має ідентифікатор, який часто називають тегом, що визначає відповідність між елементами даних в кеші і їх копіями в основній пам'яті або у віддаленому джерелі.

Коли клієнт кеша (додаток, веб-браузер, операційна система) звертається до даних, перш за все досліджується кеш. Якщо в кеші знайдено запис з ідентифікатором, що збігається з ідентифікатором затребуваного елемента даних, то використовуються елементи даних в кеші. Такий випадок називається «попаданням кеша». Якщо в кеші не знайдено запис, що містить затребуваний елемент даних, то він читається з основної пам'яті в кеш, і стає доступним для подальших звернень. Такий випадок називається «промахом кеша». Відсоток звернень до кешу, коли в ньому знайдений результат, називається рівнем влучень, або коефіцієнтом влучень в кеш.

Наприклад, веб-браузер перевіряє локальний кеш на диску на наявність локальної копії веб-сторінки, що відповідає запрошенням URL. У цьому прикладі URL - це ідентифікатор, а вміст веб-сторінки - це елементи даних.

* + 1. Алгоритми витіснення

В інформатиці під алгоритмами кешування (часто званими алгоритмами витіснення або політиками витіснення, а також «алгоритмами / політиками заміщення») розуміють оптимізацію інструкцій - алгоритми - особлива комп'ютерна програма або апаратно підтримувана структура, здатна управляти кешем інформації, що зберігається в комп'ютері. Коли кеш заповнений, алгоритм повинен вибрати, що саме потрібно видалити з нього, щоб мати можливість запису (в кеш) нової, більш актуальної інформації.

«Рівень влучень» кешу означає те, наскільки часто шукані дані виявляються в кеші. Більш ефективні політики витіснення відстежують звернення до найбільш використовуваної інформації, щоб поліпшити рівень влучень (при тому ж розмірі кешу).

«Латентність» кешу означає, наскільки швидко кеш може повернути запитані дані безпосередньо після запиту (в разі, якщо відбувається «потрапляння»). Більш швидкі стратегії витіснення зазвичай відстежують найменш використовувану інформацію - або, в разі кеша прямого відображення (direct-mapped cache), відсутність інформації, щоб знизити витрати часу на оновлення інформації.

Кожна стратегія витіснення є компромісом між рівнем влучень і латентністю.

***Алгоритм Беладі***

Найбільш ефективне правило витіснення - відкидати з кешу ту інформацію, яка не знадобиться в майбутньому найдовше. Цей оптимальний алгоритм кешування назвали алгоритмом Беладі або алгоритмом передбачення. Так як в загальному випадку неможливо передбачити, коли саме в наступний раз буде потрібно саме ця інформація, то на практиці (знову ж таки, в загальному випадку) подібна реалізація неможлива. Практичний мінімум може бути обчислений лише досвідченим шляхом, після чого можна порівняти з ним ефективність поточного алгоритму кешування.

***Витіснення давно невикористовуваних***

Least recently used (LRU): в першу чергу, витісняється невикористаний найдовше. Цей алгоритм вимагає відстеження того, що і коли використовувалося, що може виявитися досить накладно, особливо якщо потрібно проводити додаткову перевірку, щоб в цьому переконатися. Загальна реалізація цього методу вимагає збереження «біта віку» для рядків кеша і за рахунок цього відбувається відстеження найменш використаних рядків (тобто за рахунок порівняння таких бітів). У подібній реалізації, при кожному зверненні до рядка кеша змінюється «вік» всіх інших рядків.

***Найбільш недавно використовувався***

Most Recently Used (MRU): на відміну від LRU, в першу чергу витісняється останній використаний елемент. Для схем довільного доступу і циклічного сканування великих наборів даних (іноді званих схемами циклічного доступу) алгоритми кешування MRU мають більше влучень в порівнянні з LRU за рахунок їх прагнення до збереження старих даних. Алгоритми MRU найбільш корисні у випадках, коли ніж старше елемент, тим більше звернень до нього відбувається.

**Псевдо-LRU**

Псевдо-LRU (PLRU): Для кешей з великою асоціативністю (зазвичай> 4 каналів), ціна реалізації LRU стає непомірно висока. Якщо достатня схема, що майже завжди потрібно відкидати найменш використовуваний елемент, то в цьому випадку можна використовувати алгоритм PLRU, що вимагає для елемента кешу тільки один біт.

***Сегментований LRU***

Сегментований LRU: SLRU-кеш ділиться на два сегменти: пробний сегмент і захищений сегмент. Рядки в кожному сегменті впорядковані від часто використовуваних до найменш використовуваним. Дані при промахах додаються в кеш, причому в область останніх використаних елементів пробного сегмента. Дані при влучань прибираються де б вони не розташовувалися і додаються в область часто використовуваних елементів захищеного сегмента. До рядків захищеного сегмента звернення таким чином відбуваються принаймні двічі. Захищений сегмент обмежений. Таке перенесення рядки з пробного сегмента в захищений сегмент може викликати перенесення останньої використаної (LRU) рядки в захищеному сегменті в MRU-область пробного сегмента, даючи цієї лінії другий шанс бути використаною перед витісненням. Розмір захищеного сегмента - SLRU-параметр, який змінюється в залежності від схеми роботи введення-виведення. Всякий раз коли дані повинні бути витіснені з кешу, рядки запитуються з LRU-кінця пробного сегмента.

***Найменш часто використовуваний***

Least Frequently Used (LFU): LFU підраховує як часто використовується елемент. Ті елементи, звернення до яких відбуваються рідше за все, витісняються в першу чергу.

***Адаптивна заміна***

Adaptive Replacement Cache (ARC): постійно балансує між LRU і LFU, що покращує підсумковий результат.

* 1. SQL інтерфейс Java

**Java DataBase Connectivity** — прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних. JDBC — це платформо-незалежний промисловий стандарт взаємодії Java-застосунків з різноманітними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE.

В основі JDBC лежить концепція так званих драйверів, що дозволяють отримувати з'єднання з базою даних по спеціально описаному URL. Драйвери можуть завантажуватись динамічно (під час роботи програми). Завантажившись, драйвер сам реєструє себе й викликається автоматично, коли програма вимагає URL, що містить протокол, за який драйвер «відповідає».

* + 1. Компоненти інтерфейсу

1. JDBC API – забезпечує програмний доступ до реляційних даних з мови програмування Java. JDBC API також може взаємодіяти з декількома джерелами даних в розподіленому середовищі. JDBC API є частиною платформи Java, яка включає в себе Java Standard Edition і Java Enterprise Edition. JDBC 4.0 ділиться на два пакети: java.sql і javax.sql. Обидва пакети включені в платформи Java SE і Java EE.
2. JDBC Driver Manager – реєструє об'єкти, які можуть підключати Java додатки до драйверу JDBC. DriverManager традиційно є основою архітектури JDBC. Являється досить маленьким і простим.
3. JDBC Test Suite – допомагає визначити, які драйвери JDBC будуть запускати програму. Ці тести здійснюють багато важливих функцій.
4. JDBC-ODBC Bridge – це драйвер JDBC, що реалізує операції JDBC шляхом трансляції їх в операції ODBC. З точки зору ODBC - це звичайна програма. Міст таким чином надає JDBC-інтерфейс до будь-яких СУБД, для яких доступний ODBC-драйвер. Міст реалізований у вигляді пакету sun.jdbc.odbc і містить бібліотеку для доступу до ODBC. Цей міст - спільна розробка Intersolv і JavaSoft.
   * 1. Головні інтерфейси

* java.sql.DriverManager забезпечує завантаження драйверів і створення нових з'єднань (connection) з базою даних; це основний інтерфейс JDBC, який визначає коректний вибір і ініціалізацію драйвера для даної СУБД в даних умовах;
* java.sql.Connection визначає характеристики і стан з'єднання з БД; крім того, він надає функціональність для контролю транзакцій і рівня їх ізольованості;
* java.sql.Statement виконує функції контейнера по відношенню до SQL-виразу; при цьому під виразом розуміється не тільки сам текст запиту, а й такі характеристики, як параметри і стан виразу;
* java.sql.ResultSet надає доступ до набору рядків, отриманого в результаті виконання даного SQL-виразу.

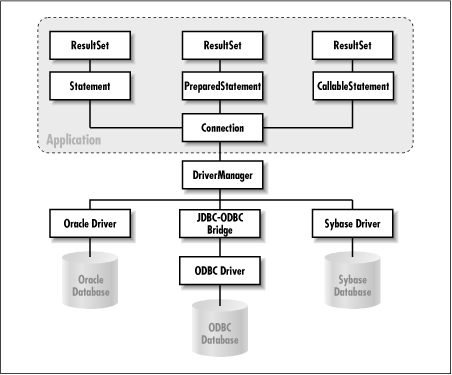


Рис 1. Основні інтерфейси JDBC

Інтерфейс java.sql.Statement виступає в якості предка для інших двох важливих інтерфейсів: java.sql.PreparedStatement і java.sql.CallableStatement, перший з яких призначений для виконання прекомпільованих SQL-виразів, другий - для виконання викликів збережених процедур. Відповідно Statement виконує звичайні (статичні) SQL-запити, а зазначені два спадкоємця працюють з параметризованими SQL-виразами.

* + 1. Архітектури взаємодії з JDBC

JDBC API підтримує обидві архітектури доступу до даних, як дворівневу так і трирівневу.



Рис 2. Дворівнева архітектура для доступу до даних

У дворівневій моделі Java аплет або деякий додаток безпосередньо взаємодіє з джерелом даних. Для цього потрібен JDBC драйвер, який може працювати з конкретним джерелом даних до якого здійснюється доступ. Команди користувача передаються до конкретного джерела даних і, в результаті опрацювання даної команди, отримується результат виконання даної команди. Джерело даних може бути розташоване на іншій машині, сервері, до якого користувач підключений через мережу.



Рис 3. Трирівнева архітектура для доступу до даних

На рис. 3 представлена ​​діаграма взаємодії Java-коду і баз даних в трирівневій (точніше, багаторівневій N-tier) моделі. Як ви бачите, в якості основних способів взаємодії між Java-додатками і серверами середньої ланки використовуються механізми RPC (виклики віддалених процедур) і CORBA. У разі CORBA, взагалі кажучи, крім ланки бізнес-логіки присутній ORB (брокер об'єктних запитів, брокер об'єктних заявок). Що стосується самих серверів додатків, то Java (зокрема, JDBC) ніяких спеціальних вимог до них не пред'являє, як і в випадку з серверами баз даних.

* 1. Огляд існуючих рішень
     1. EHCACHE

Ehcache - широко поширений Java-розподілений кеш з відкритим вихідним кодом для кешування загального призначення, використовується як в Java EE так і в фреймворках. Ehcache доступний на умовах ліцензії з відкритим вихідним кодом Apache і активно підтримується в даний момент багатьма дистрібьюторами.

Ehcache був спочатку розроблений Грегом Люком, починаючи з 2003 року. У 2009 році проект був куплений компанією Terracotta, що передбачає оплачувану підтримку.

Програмне забезпечення залишається відкритим вихідним кодом, але деякі основні функціональні можливості (Fast Restartability Consistency й інші) доступні тільки в комерційних продуктах, таких як Enterprise Ehcache і BigMemory, які не являються на даний момент додатками з відкритим вихідним кодом.

У березні 2011 року Фонд Вікімедіа оголосив, що буде використовувати Ehcache для поліпшення продуктивності своїх вікі-проектів. Однак вони швидко відмовилися від даного продукту після перевірки виявлених проблем і розумінням того, що виправлення даних проблем буде коштувати немалу частку грошей для компанії.

EHCache може реалізовувати кілька різних стратегій кешування, наприклад - LFR (кешування виходячи з частоти використання) або FIFO, також можна контролювати кеш на рівні окремих об'єктів (задаючи схему інвалідаціі об'єктів в кеші - за часом життя або часу простою).

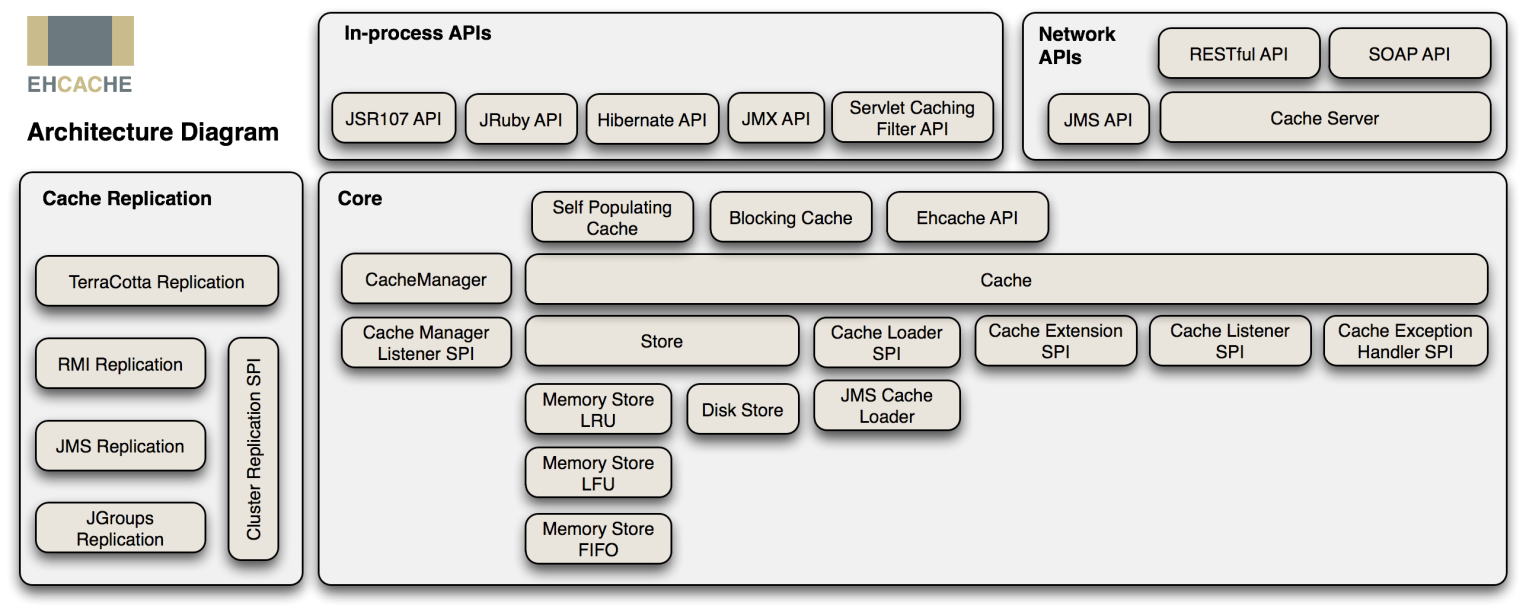


Рис. 4. Архітектура EHCACHE

Додаток підтримує стандартний протокол роботи з кешем - JSR107 JCACHE.

Частина функцій EHCACHE реалізуються через сторонні компоненти (наприклад, розподілені кеші). Розподілене кешування може застосовуватися для забезпечення роботи на кластері або в інших розподілених схемах, використовується вбудований (з версії 1.2) механізм RMI на основі протоколу TCP. Синхронізація та / або реплікація кешів між вузлами може бути як загальною, так і локальною, для окремих кешів за своєю схемою, асинхронною або синхронною.

Однією з унікальних функцій ehcache є «персистентний кеш», який дозволяє зберігати стан об'єктів навіть після перезавантаження віртуальної java-машини. Крім цього, система кешування підтримує консоль управління JMX, що дозволяє інтегрувати її в систему управління будь-яким додатком, використовуючи тільки стандартні протоколи і можливості.

*Переваги:*

* Підтримка різних стратегій кешування;
* Підтримка механізму RMI;
* Наявність персистентного кешу;
* Підтримка консолі JMX;
* Наявність розподіленого кешу;
* Підтримка сучасних фреймворків;
* Підтримка стандартного протоколу роботи з кешем;
* Підтримка кешування web-частини;
* Має досить інтуїтивний інтерфейс.

*Недоліки:*

* Надмірна функціональність;
* Об’єктно-орієнтоване кешування;
* Не підтримує ранні версії Java;
* Велика частина функціональності являється не безкоштовною;
* Неефективна і складна інтеграція з стандартним SQL інтерфейсом Java;
* Різні версії додатку мають велику кількість помилок в вихідному коді;
* Залежність від сторонніх бібліотек;
* Фреймворк-орієнтована розробка;
  + 1. OSCache

OSCache є високопродуктивним і широко використовуваним рішенням кешування в J2EE. OSCache розробляється і контролюється проектом OpenSymphony та є частиною проекту OpenSymphony. Вихідний код доступний під так званою OpenSymphony ліцензією на програмне забезпечення, яке засноване на ліцензії Apache.

OSCache може використовуватися в якості автономного рішення кешування для будь-якої програми Java.

Бібліотека тегів OSCache і фільтр кешування для динамічного довічного контенту, такого як PDF-файли або зображення, вимагає Servlet 2.3 / JSP 1.2 сумісного контейнера.

OSCache вирішує основні проблеми для динамічних веб-сайтів:

Кешування динамічного контенту - динамічні оновлення в тій чи іншій формі часто повинні бути виконані під час кожного запиту, але іноді, контент сторінки не змінюється кожен запит. Кешування всієї сторінки не допоможе, тому що окремі розділи сторінки можуть змінюватись кожен запит. OSCache вирішує цю проблему, надаючи функціонал для кешування розділів сторінок JSP.

Кешування бінарних файлів - завантажені зображення, бінарізовані і PDF-файли можуть створювати проблеми з точки зору навантаження на сервер. OSCache вирішує цю проблему через Servlet 2.3 CachingFilter який може кешувати будь-які URI (наприклад, всієї сторінки або зображення).

Стійкість до помилок - якщо одна помилка виникає десь на динамічної сторінці, швидше за все, вся сторінка буде повернута як помилка, навіть якщо 95% сторінки виконалось без помилок. OSCache вирішує цю проблему, дозволяючи кешування в разі помилки, а потім, відповідним чином, поставляється повідомлення про помилку

*Переваги:*

* Наявність персистентного кешу;
* Підтримка кластеризації;
* Підтримка фреймворку Hibernate;
* Підтримка кешування web-частини;
* Можливість кешувати звичайні Java об’єкти.

*Недоліки:*

* Веб-орієнтоване кешування;
* Частина функціональності являється не безкоштовною;
* Немає інтеграції з SQL інтерфейсом Java;
* Залежність від сторонніх бібліотек;
* Фреймворк-орієнтована підтримка;
* Обмеження яке вимагає наявність Servlet 2.3 / JSP 1.2 сумісного контейнера;
* Відсутність широкого розповсюдження у комп’ютеризованих додатках.
  + 1. SwarmCache

SwarmCache є простим, але ефективним розподіленим кешем. Для ефективної взаємодії з будь-яким числом хостів в локальній мережі він використовує IP Multicast. Спеціально розроблений для роботи у кластерних середовищах для веб-додатків що використовують бази даних. Такі програми, як правило, мають більшу кількість операцій читання ніж операцій запису, що дозволяє SwarmCache збільшити продуктивність додатку. SwarmCache використовує JavaGroups для управління і комунікації з розподіленої кеш-пам'яттю.

Для даного кешу існують інтерфейси для взаємодії з такими популярними фреймворками як Hibernate та JPOX.

Концепція SwarmCache досить проста. Кожен сервер ініціалізує свій власний менеджер. Для кожного типу об'єкта що сервер бажає закешувати, він створює кеш і додає його менеджеру. Менеджер приєднується до multicast групи і обмінюється даними з іншими менеджерами в групі. Всякий раз, коли об'єкт віддаляється з кешу, менеджер сповіщає всі інші менеджери в групі. Ці менеджери гарантують, що об'єкт буде видалений з їх відповідних кешів. Результатом є те, що сервери не будуть мати в своєму кеші застарілої версії об'єкта, який був оновлений або видалений на іншому сервері.

*Переваги:*

* Підтримка різних стратегій кешування;
* Наявність IP multicast;
* Розподілений кеш;
* Підтримка декількох сучасних фреймворків;
* Має досить інтуїтивний інтерфейс.

*Недоліки:*

* Об’єктно-орієнтоване кешування;
* Відсутня інтеграція з стандартним SQL інтерфейсом Java;
* Залежність від сторонніх бібліотек;
* Фреймворк-орієнтована розробка;
* Складний у налаштуванні;
* Відсутність широкого розповсюдження у комп’ютеризованих додатках;
* Відсутність транзакційності;
* Наявність тільки розподіленого кешування.
  + 1. JBoss TreeCache

JBoss TreeCache – деревовидний, розподілений, транзакційний кеш який є основою для багатьох фундаментальних служб кластеризації сервера додатків JBoss. Може бути використаний в якості окремого кешу або навіть в якості об’єкто-орієнтованого сховища даних. Може бути вбудований в інші J2EE-сумісні сервера, такі як BEA WebLogic або IBM WebSphere, контейнери сервлетів, таких як Tomcat, або навіть в Java додатки, які не виконуються всередині сервера додатків.

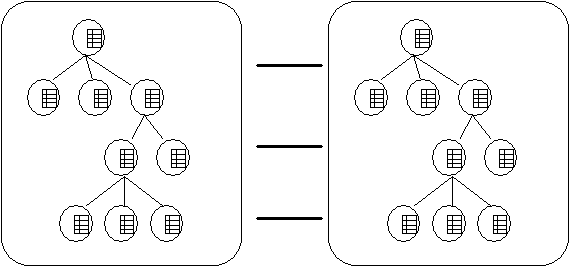


Рис. 5. Схематична архітектура TreeCache

TreeCache є дерево з вузлами. Кожен вузол має ім'я і нуль або більше дітей. Вузол може мати тільки одного батька; в даний час немає підтримки для графів. Вузол може бути досягнутий шляхом переходу від кореня рекурсивно через дитину, поки запитуваний вузол не буде знайдений. Також шукома інформація може бути знайдена через повне ім’я. TreeCache може мати кілька коренів, що дозволяє для ряду різних дерев бути присутніми в одному екземплярі кеша. Також може бути локальним або розподіленим кешем.

*Переваги:*

* Підтримка різних стратегій кешування;
* Наявність персистентного кешу;
* Наявність розподіленого кешу;
* Наявність транзакційності;
* Наявність ліцензії що забезпечує підтримку виробника.

*Недоліки:*

* Надмірна функціональність;
* Необхідність придбання дуже дорогої ліцензії;
* Орієнтованість на сервера додатків;
* Відсутність інтеграції з стандартним SQL інтерфейсом Java;
* Залежність від сторонніх бібліотек;
* Складність у налагодженні;
* Необхідність у потужних серверах;
* Не інтуїтивний інтерфейс;
* Складність впровадження.
  1. Постановка задачі

Метою роботи є створення кешу малозмінних даних на основі SQL інтерфейсу мови програмування Java який повинен відповідати наступним вимогам:

* Простота у використанні;
* Інтуїтивно зрозумілий інтерфейс;
* Простота у налагодженні;
* Можливість використовувати кеш на будь-якій сучасній платформі;
* Кеш повинен забезпечувати компактне збереження даних;
* Кеш повинен зменшити навантаження на БД;
* Кеш повинен зменшити час отримання малозмінної інформації у комп’ютеризованій системі.

Об’єктом дослідження є алгоритми кешування даних, а предметом дослідження – структури даних в котрих можна зберігати дані.

Процес створення кешу малозмінних даних комп’ютеризованої системи можна розділити на три складових – розробка структури даних для компактного збереження SQL запитів, розробка алгоритму кешування результатів даних SQL запитів та тестування і налагодження кешу. Отже задачі, що вирішуються в роботі:

1. Аналіз і реалізація структури даних для компактного збереження SQL запитів;
2. Аналіз і реалізація алгоритму кешування;
3. Розробка кешу малозмінних даних на основі SQL інтерфейсу мови програмування Java що відповідає вимогам, зазначеним вище.

## Висновки по розділу

У першому розділі проведено аналіз існуючих рішень та предметної області, проаналізовані основні сучасні засоби кешування даних у мові програмування Java, визначено переваги та недоліки цих рішень. В результаті проведеного аналізу сформульована постановка задачі, наведене призначення, цілі та задачі розробки.