ЛАБОРАТОРНА РОБОТА 6

Тема: Технологія Enterprise JavaBeans (EJB)

Специфікація Enterprise JavaBeans (EJB) пережила три реалізації. Перша версія EJB представила індустрії інноваційні рішення, але обмеження функціональних можливостей призвели до нових вимог. У другій версії технологія EJB надала промисловим розробникам: підтримку віддалених взаємодій, механізми управління транзакціями, засоби забезпечення безпеки, обробки та зберігання інформації та веб-служби. Всі ці механізми є великоваговими, що вимагають від розробників основну увагу приділяти особливостям взаємодій із самою інфраструктурою, ніж реалізації бізнес-логіки додатків. Версія EJB 3 стала простішою і легковажнішою, зберігши свою міць, а компоненти EJB тепер можуть бути простими об'єктами Java (Plain Old Java Objects, POJO).

По суті, EJB представляє колекцію фіксованих рішень типових проблем, що виникають при розробці серверних програм, а також перевірену часом схему реалізації серверних компонентів. Ці фіксовані рішення або служби надаються контейнером EJB. Для доступу до цих служб необхідно створити спеціалізовані компоненти, використовуючи декларативні та програмні EJB API, та розгорнути їх у контейнері. Далі серверні компоненти EJB можна використовувати для побудови прикладних компонентів. Компонент EJB 3 – це POJO з деякими спеціальними можливостями, які залишаються невидимими, доки вони не стануть необхідні та не відволікають від головної ролі компонента.

Для того, щоб скористатися службами EJB, компонент повинен бути оголошений з типом, що розпізнається фреймворком EJB. EJB розпізнає два конкретні типи компонентів: сеансові компоненти (session beans) та компоненти, керовані повідомленнями (message-driven beans). Сеансові компоненти поділяються на сеансові компоненти без збереження стану (stateless session beans), сеансові компоненти зі збереженням стану (stateful session beans) та компоненти-одинаки (синглтони – singletons). Компоненти кожного типу мають певне призначення, область видимості, стан, життєвий цикл та особливості використання на рівні прикладної логіки. Усі компоненти EJB є керованими. Керовані компоненти по суті є звичайними Java-об'єктами в середовищі Java EE. Механізм управління контекстами та впровадження залежностей (Contexts and Dependency Injection, CDI) дозволяє здійснювати впровадження залежностей у будь-які керовані компоненти, включаючи компоненти EJB.

З точки зору розробки програмного забезпечення, фіксовані служби є найбільш цінною частиною EJB. Деякі служби автоматично підключаються до зареєстрованих компонентів, оскільки спеціально призначені для використання компонентами на рівні прикладної логіки. До цих служб входять: використання залежностей, транзакції, підтримка багатопоточної моделі виконання та організація пулів. У більшості випадків для підключення служб розробник має явно оголосити про свої наміри через анотації/XML або звернення до EJB API під час виконання. До таких служб належать: підтримка безпеки, планування, асинхронна обробка, віддалені взаємодії та веб-служби.

Корпоративні програми проектуються для вирішення завдань, до яких пред'являється певна кількість подібних вимог: повинні мати деякий інтерфейс користувача, реалізувати прикладні процеси, модель предметної області і зберігати інформацію в базі даних. Так як ці вимоги є загальними, з'являється можливість дотримуватися узагальненої архітектури, або принципів проектування, відомим як шаблони проектування.

При розробці серверних програм часто використовуються багаторівневі архітектури, в яких компоненти групуються в рівні (або шари). Кожен рівень додатка служить чітко визначеним цілям і передає (делегує) виконання операцій наступного рівня, розташованого під нею. EJB відповідає моделі спеціалізованих компонентів, які призначено для вирішення завдання, що властиві певному рівню багаторівневої архітектурі. Існує два основні різновиди багаторівневих архітектур: традиційна чотирирівнева архітектура та проблемно-орієнтована архітектура (Domain-Driven Design, DDD).

На рис. 1 представлено традиційну чотирирівневу архітектуру, яка має велику популярність. У цій архітектурі рівень представлення відповідає за відображення графічного інтерфейсу користувача (Graphical User Interface, GUI) та обробку введення користувача. Часто цей рівень реалізується із застосуванням браузера або окремої програми. Рівень представлення передає всі запити до рівня прикладної логіки, який є ядром програми і містить основну логіку обробки, що визначає бізнес-правила. На цьому рівні знаходяться компоненти, що виконують різні прикладні операції, наприклад облік, пошук, впорядкування даних, обслуговування облікових записів, тощо. Рівень прикладної логіки отримує та зберігає дані у базі, при цьому використовується рівень зберігання. Рівень зберігання забезпечує високорівневі, об'єктно орієнтовані абстракції над рівнем бази даних. Рівень бази даних зазвичай включає систему управління реляційними базами даних (Relational Database Management System, RDBMS).

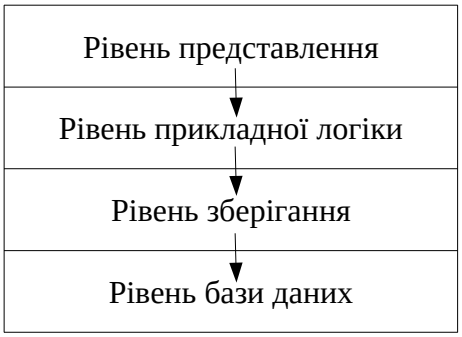

Рис.1. Чотирирівнева архітектура додатку.

Технологія EJB забезпечує підтримку компонентів прикладного рівня (рис.2). Усі сервіси є незалежними один від одного, тому їх вибір для використання у додатку є довільним.

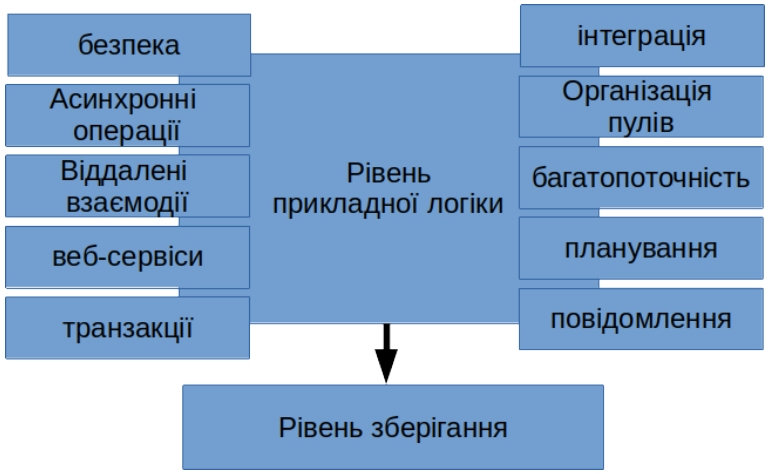

Рис.2. Сервіси компонентів EJB 3 на прикладному рівні.

У типовій системі на основі Java EE, на рівні представлення використовуються JSF та CDI, EJB використовується на прикладному рівні, а на рівні зберігання даних використовуються JPA та CDI.

До недоліків традиційної чотирирівневої архітектури можна віднести її орієнтацію на моделювання прикладної обробки, а не предметної області. Тому рівень прикладної логіки часто нагадує додаток баз даних, написаний у процедурному стилі, а не об'єктно-орієнтований додаток. Оскільки компоненти рівня зберігання є простими сховищами даних, вони більше нагадують записи у базі даних, а не сутності ООП. Проблемно-орієнтована архітектура (Domain-Driven Design, DDD) є альтернативним рішенням, яке не має цих недоліків. Архітектура DDD підкреслює, що об'єкти предметної області не повинні бути простою заміною записів бази даних, і повинні містити прикладну логіку. Ці об'єкти можуть бути реалізовані як сутність у JPA.

Наприклад, об'єкт Catalog у додатку електронної торгівлі, дотримуючись архітектури DDD, крім зберігання даних з таблиці каталогу товарів бази даних, може забезпечувати аналіз та не повертати записи товарів каталогу, які тимчасово відсутні на складі. Сутності JPA є простими об'єктами Java (POJO) і також підтримують успадкування та поліморфізм.

На рис. 3 показано, як виглядає проблемно орієнтована архітектура. Рівень представлення відповідає за інтерфейс користувача та взаємодії з рівнем сервісу/додатку. Рівень сервісу/додатку забезпечує взаємодію між рівнем представлення та предметним рівнем, як правило, є тонким та легковажним. Предметний рівень – це складний комплекс компонентів, що відтворюють модель прикладних даних, які складаються із сутностей, об'єктів значень, агрегатів, фабрик та репозиторіїв. Рівень інфраструктури відповідає базі даних чи іншій технології зберігання даних.

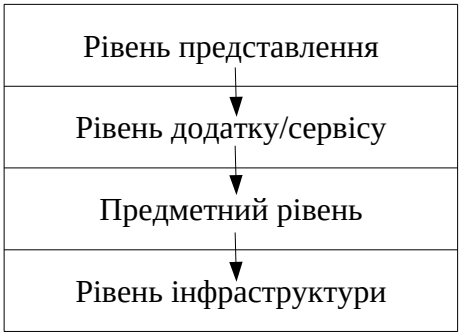

Рис.3. Проблемно-орієнтована архітектура

Enterprise JavaBeans (EJB) - це керований компонент, що належить стороні сервера для модульного конструювання програми рівня підприємства. Компонент означає, що EJB розповсюджуються в бінарному форматі і можуть налаштовуватися, так що їх можна використовувати при створенні клієнтської програми. Термінологічна відмінність між компонентами та екземплярами:

- термін Enterprise JavaBeans використовується для вказівки типу компонента - наприклад, EJB, що представляє банківський рахунок;
- термін "екземпляр EJB" використовується для вказівки на об'єкт, який представляє специфічний банківський рахунок з унікальним номером рахунку.

"Сторона сервера" означає, що об'єкт EJB розміщується в процесі на тому самому сервері, а не на клієнтській машині. EBJ може надавати свій інтерфейс як віддалено, так і локально. Якщо EJB надає віддалений перегляд свого інтерфейсу, всі виклики методів між клієнтським кодом і віддаленим екземпляром відбуваються через протокол віддаленого виклику процедур, такого як RMI-IIOP (віддалений виклик методу поверх з використанням протоколу зв'язку CORBA). Якщо EJB представляє локальний перегляд, клієнтський код має бути в тому ж процесі, що й EJB об'єкт, а всі виклики є прямими методами.

Здатність виконувати віддалений виклик процедур передбачає наявність двох фундаментальних елементів архітектури: іменовані сервіси та RPC проксі. Іменовані сервіси – це мережеві сервіси, які використовують для знаходження гетерогенних ресурсів мережі. RPC проксі - це програмно згенерований Java клас, створений на стороні клієнта, який представляє такий самий інтерфейс, що і специфічний віддалений компонент. Так як він надає такий самий інтерфейс, то для клієнтів такий проксі відіграє роль віддаленого екземпляра, а функціонально - проксі делегує всі виклики віддаленому компоненту, що реалізує мережевий код, складність якого клієнту не є доступною. У EJB використовується аналогічна концепція іменованих сервісів та RPC проксі.

Однією з найважливіших властивостей EJB є здатність бути керованими. До активного в пам'яті об'єкту EJB має розширений доступ процес, що його веде. Цей процес викликається EJB контейнером і є стандартизованим середовищем часу виконання, яке надає функціональність, що управляє. Специфікація EJB стандартизує сервіси, що надаються контейнером EJB,

функціональність яких реалізується через певних постачальників, наприклад, WildFly або GlassFish та ін.

Усі контейнери EJB мають такі сервіси:

- постійність об'єкта;
- декларативний контроль безпеки;
- декларативний контроль транзакцій;
- управління конкуруванням (паралельністю);
- управління масштабуванням.

Постійність об'єкта означає, що можна відобразити стан екземпляру EJB на дані в деякому постійному сховищі, наприклад, в реляційній базі даних. Контейнер зберігає стан у постійному сховищі, яке синхронізоване зі станом EJB екземпляра, навіть коли цей об'єкт використовується та модифікується паралельно кількома клієнтами.

Декларативний контроль безпеки дозволяє встановлювати контейнеру перевірку, що клієнт, який викликає певний метод для деякого EJB, був авторизований і належить до ролі, яка очікується. Якщо це не так – буде згенеровано виключення і метод не буде виконано. Вигода від декларативного контролю безпеки в тому, що немає необхідності створювати код для будь-якої логіки безпеки в EJB - просто визначаються ролі та повідомляється контейнеру, яким ролям дозволяється викликати певні методи. Так як немає жорстко закодованої логіки безпеки в EJB, то легко зробити зміни відповідно до вимог безпеки без перекомпіляції.

Декларативний контроль транзакцій є аналогічним механізмом. Контейнер повідомляється про певні дії і декларується, що має відбуватися в термінах розмежування транзакцій, коли викликається певний метод. Наприклад, можна вказати контейнеру почати нову транзакцію, коли викликається певний метод, або можна вказати використовувати існуючу транзакцію та відхилити виклик методу, якщо один із методів ще не активний (транзакція проходить по ланцюжку викликів методів). Контейнер автоматично підтверджуватиме транзакцію, коли метод, що спричинив запуск транзакції, завершиться коректно, або він відкотить транзакцію, якщо буде перехоплено виключення, що з'явився в той час, коли транзакція була активною. Користь від декларативного управління транзакціями в тому, що немає необхідності поміщати логіку транзакцій у вихідний код EJB, який створюється, що не тільки спрощує розробку, але також полегшує зміну поведінки транзакції EJB без перекомпіляції. Можливість здійснювати декларативний контроль над логікою безпеки та транзакцій є фундаментальною вимогою до компонентної моделі, тому що це дозволяє розробникам, які не мають вихідного коду, адаптувати компоненти до додатку, що будується за допомогою цих компонентів.

Управління конкуруванням синхронізує паралельні виклики методів, що надходять від різних віддалених клієнтів, і направляє їх до одного і того ж EJB об'єкту. На практиці, це гарантує, що компонент може бути безпечно використаний у наслідуваному багатопотоковому середовищі сервера додатків, що є досить корисною властивістю.

Управління масштабуванням пов'язане з проблемою збільшення ресурсів, що виділяються (з'єднання з базою даних, нитки (threads), сокети, черга повідомлень тощо), яка виникає при зростанні кількості клієнтів, що одночасно працюють. Наприклад, при найпростішому підході до цієї проблеми, якщо кількість клієнтів, що одночасно працюють, збільшується, то пропорційно буде збільшуватися кількість EJB об'єктів у пам'яті і кількість відкритих з'єднань з базою даних. Це може бути занадто великим значенням для кількості наявної пам'яті, так і серйозним навантаженням на сервер бази даних. EJB контейнер може вирішити цю проблему, надавши об'єднані (pooling) екземпляри EJB та об'єднані (pooling) з'єднання з базою даних. Таким чином, EJB контейнер зберігатиме лише обмежену кількість екземплярів або з'єднань, що живуть у пам'яті, і присвоюватиме їм різних клієнтів тільки в той час, коли клієнт дійсно потребуватиме цього. Як результат, ресурси виділятимуться для клієнта, а не для EJB об'єктів.

Завдання

1. Вивчить та реалізуйте приклад додатку Java EE з використанням технології EJB3, який наводиться за адресою https://netbeans.apache.org/kb/docs/javaee/javaee-entapp-ejb.html.

2. Наведіть схему взаємодії компонент у додатку.

3. Розширте можливості прикладу: додайте список певних розділів новин (наприклад, «політика», «розваги», «наука», тощо); забезпечте показ новин за розділами.

4. Підготуйте звіт.