S04-2 Elosztott alkalmazások készítése

Tartalom

1. Többrétegű architektúra, elosztott szerveroldal
2. Kommunikációs eszközök: távolieljárás-hívás és üzenet alapú infrastruktúra (point-to-point és publish-subscribe modell)
3. Névszolgáltatás
4. Enterprise JavaBeans komponensmodell, EJB-k fajtái
5. Beanek életciklusa
6. Dependency injection
7. Elosztott és globális tranzakciók

Többrétegű architektúra

- 3-rétegű alkalmazás (fizikailag elkülönülnek):
  - kliens (megjelenítés)
    - pl. web kliensek, GUI alkalmazások
  - üzleti logika
    - általában EJB segítségével
  - adat (Enterprise Information System)
    - pl. adatbázisok
- Középső réteg elkülönítésének előnyei:
  - egyszerűbb (akár többféle) kliens
  - biztonságos, megbízható
  - könnyebb karbantartás, fejlesztés
- 4-rétegű alkalmazásokban megjelenik a web-réteg (kliens és üzleti logika között)
  - megjelenítési logika megvalósítása
  - még egyszerűbb kliens, skálázhatóság
- Az egyes rétegek komponensekből épülnek fel, fajtái: kliens, szerver, web
- Minden komponens egy megfelelő tárolóban fut, és az ezek által nyújtott szolgáltatások (middleware services) megoldást biztosítanak általános problémákra
  - Távoli eljáráshívás
  - Szálkezelés
  - Terheléskiegyenlítés
  - Átlátszó hibakezelés
  - Perzisztencia
  - Tranzakciókezelés
  - Objektumok életciklusa
  - Aszinkron üzenetkezelés
  - Biztonság
  - Resource-pooling

  - Autentikáció
  - Authorizáció
  - Caching
  - Clustering
  - stb.

Távolieljárás-hívás

- Remote Procedure Call
  - egy másik gépen található függvény hívása
  - a socket kommunikáció szintaxisát fedi el
  - az objektumok érték szerint, szerializáció segítségével adódnak át
  - a kliensben és a szerverben is egy stub objektum végzi a kommunikációt
- Remote Method Invocation (RMI)
  - RPC-alapú kommunikáció egy megvalósítása
  - JVM-ek közötti kommunikáció
- Java Remote Method Protocol (JRMP)
  - java.rmi csomagban található RMI megvalósítás (szerializáció, TCP/IP)
  - a bejövő hívásokat külön szálon kezeli, ezért a távoli objektumokat érdemes thread-safe módon megvalósítani
  - névszolgáltatást biztosít a kommunikációs partnerek megtalálásához
- RMI over IIOP
  - RMI egy másik megvalósítása (javax.rmi)
  - Internet Inter-ORB Protocol használata kommunikációra
    - CORBA használja nyelv- és vendorfüggetlen együttműködésre
  - kevésbé rugalmas, mint a JRMP
  - EJB is ezt használja

Üzenet alapú infrastruktúra

- Komponensek közötti kommunikáció
- A küldő és a fogadó nincs közvetlen kapcsolatban → köztes szolgáltató (provider)
  - aszinkron üzenetküldés
  - akár különböző protokollok használata
- Java Message Service (JMS): specifikálja, hogy kommunikálhatnak Java programok
- Point-to-point (P2P)
  - két végpont közötti üzenetküldés
  - a küldő egy message queue-ra ír, a fogadó ebből veszi ki az üzeneteket
- Publish-subscribe
  - a fogadók feliratkoznak topicokra, a küldők ezekre publikálnak

  - a topicok minden feliratkozónak továbbítják a közzétett üzenetet
  - az offline fogadók csak durable subscription esetén kapják meg a kihagyott üzeneteket

Névszolgáltatás

- Entitások (fájl, számítógép, objektum, stb.) megtalálása név alapján
- Műveletek:
  - név rögzítése a szolgáltatásban
  - keresés név alapján
  - rögzített nevek karbantartása
- Pl. DNS, fájlrendszer
- Java Naming and Directory Interface (JNDI)
  - egységes interfész név- és könyvtárszolgáltatásokhoz
  - nem kell ismernünk a szolgáltatás konkrét megvalósítását
  - könyvtárszolgáltatás
    - a névszolgáltatás kiegészítése
    - attribútumok rendelhetők az entitásokhoz
  - pl.:
    ```
    Context ctx = new InitialContext();
    DataSource ds = (DataSource) ctx.lookup("jdbc/DB");
    ```

Enterprise JavaBeans komponensmodell

- Szerveroldali komponensmodell obj.-orientált elosztott alkalmazások készítéséhez
- EJB tulajdonságai:
  - üzleti logikát tartalmaz
  - az EJB komponensek EJB tárolókban hajtódnak végre (szolgáltatások)
  - a tárolóján keresztül férhetünk hozzá egy komponenshez
  - a komponensek fordítás után is konfigurálhatók

Session Bean

- a kliens szerveroldali kiterjesztése
- függvényhívásokon keresztül nyújt szolgáltatásokat
- session = hívások sorozata
- a tároló feladata, hogy nyilvántartsa a kliens-bean kapcsolatokat
- `@Stateless`
  - a belső állapota kliens-független, a hívások egymástól függetlenek
  - nem garantált, hogy a kliens végig ugyan azzal a beannel kommunikál
  - életciklus:
    - a példányok egy pool-ban tárolódnak (tétlen állapot)

        * ha egy kliens hív egy függvényt, akkor egy tetszőleges példány kikerül a poolból, végrehajtja azt, majd visszakerül
        * rendszer kivételnél megsemmisül a bean példány
- `@Stateful`
  - minden klienshez tartozik egy bean példány → session függő adatok
  - egy bean állapota elmenthető és visszatölthető
  - életciklus:
    * a kliens kér egy referenciát → hozzátársít a tároló egy beant
    * a függvényhívásokat ugyan ez a bean hajtja végre, közben megőrzi az állapotát
    * a session a bean eltávolításával ér véget
    * kivétel vagy timeout esetén a bean megsemmisül
- `@Singleton`
  - 1 példány / alkalmazás / JVM
  - lehet állapota
  - a kliensek közösen használják → figyelni kell a konkurencia-kezelésre
  - életciklus:
    * `@Startup`-al annotálva a rendszer indulásakor jön létre (eager)
    * a rendszer leállásakor semmisül meg
    * rendszer kivétel esetén nem semmisül meg

Message-Driven Bean

- aszinkron függvényhívások JMS üzenetek segítségével
- `void` vagy `Future` visszatérési értékük kell, hogy legyen
- életciklus:
  - egy pool-ban tárolódnak a példányok
  - két állapota van: nem létező és fogadásra kész

Dependency injection

- Inversion of Control
  - egy szolgáltatás használatakor elég annak interfészét ismerni
  - a tényleges szolgáltatásokat futási időben kapcsoljuk hozzá az alkalmazáshoz
  - leggyakoribb megvalósítása: Service Locator (JNDI) és dependency injection
- segítségével az egyes modulok könnyen cserélhetők
- `@EJB` annotáció: a tároló fogja beinjektálni az interfész megvalósítását
  - A megvalósítás csak EJB lehet, POJO nem
  - Nem támogat scope-okat
  - pl.: `@EJB(lookup="java:global/ConverterBean!Converter")`

Elosztott és globális tranzakciók

- adatbázisok konkurens kezelésekor szükség van szinkronizációra a konzisztens állapot megőrzéséhez
- tranzakció: adat lekérdezések és módosítások egysége
- ACID tulajdonságok
  - Atomicitás (Atomicity): vagy teljesen végrehajtódik (commit) vagy semennyire (rollback)
  - Konzisztencia (Consistency): konzisztens állapotból konzisztens állapotba vezet
  - Izoláció (Isolation): konkurens rendszerben a tranzakciók nem interferálnak
  - Tartósság (Durability): sikeres tranzakció eredménye tartós
- a tranzakciók kezelése egy köztes szolgáltatás, a transaction manager segítségével történik
- lokális tranzakció
  - a transaction manager a resource manager része
  - műveletek csak azon az egy erőforráson végezhetők
- globális tranzakció
  - különálló transaction manager
  - egy tranzakción belül több különálló erőforrás is használható
  - a fizikailag elosztott erőforrások koordinálása a 2-fázisú commit protokoll segítségével valósul meg
    1. fázis: minden resource managert megkérdez, hogy commitolhat-e
    2. fázis: mindegyik
       * commitol, ha az össze válasz igen volt
       * roll back, különben
  - Java Transaction API segítségével valósítható meg