1. Bevezető

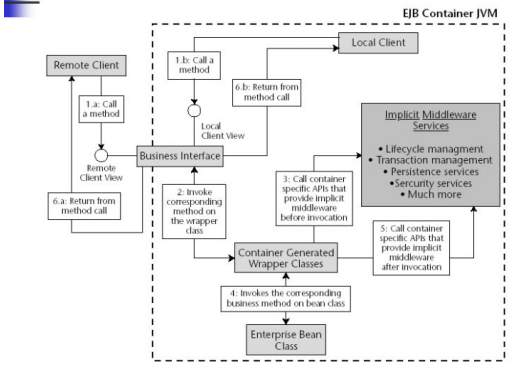
- **JavaEE def:**
  - **JavaEE platform egy architektúra vállalati méret alkalmazások fejlesztésére, a Java nyelv és internetes technológiák felhasználásával**
  - **szolgáltatásokat** definiál, keretrendszert nyújt
  - **Szolgáltatások:**
    - Többszálúság
    - Tranzakciókezelés
    - Biztonság
    - Perzisztencia
    - Névszolgáltatás
    - Objektumok életciklusának kezelése
    - Távoli metódushívás
    - Aszinkron üzenetkezelés
    - Skálázhatóság
    - Terhelés-kiegyenlítés
  - **Java EE apik:**
    - JPA JAva persistence Api: objektum-relációs leképezés
    - EJB: üzleti logikai komponenstek
    - JMS: aszonkron üzenetküldés
    - JTA Java transaction Api: tranzakciókezelés
    - Context and Dependency Injection for Java: CDI, függőséginjektálás
    - Java Authentication and Autharization Service: JAAS biztonság
  - Webes technológiák:
    - Java Servlet
    - JavaServerPages
    - JavaServerFaces
  - Webszolgáltatások:
    - **JAX-WS:** Java api for xml-based web services, SAOP alapú
    - **JAX-RS:** Web services REST tipúsú webszolgátatásokhoz
  - Hogyan fejlesztik őket?
    - JCP Java Community Process,JSR Java Specification Request-ek formájában definiálva
    - JavaEE API--kat megvalósító szfotvertermék: **alkalmazásszerver**
- **Tipikus felbontás:**
  - **Adat réteg** -
  - **Üzleti logikai réteg**
  - **Kliens** vastag (desktop alkalmazás, vékonykleins: böngésző)
    - **Web réteg** - erre csatlakoznak a vékony kliensek
- **JavaEE Konténerek**
  - Applet konténer / Application Client konténer
  - Web konténer
  - EJB konténer -> DB
- **JAvaEE komponensek**
  - Hordozható, lazán csatoltak, osztályok, interfészek kontéenerben futnak

    - **interfészen keresztül érik el egymást**
    - közé ékelhető proxy!
    - Lehetnek elosztottak
    - Hleyátlátszóság
- Profilok
    - Full profil vs. Web profile, kevesebb van benne.
        - pl.webprofilban nincs webservice, EJB egy része, JMS, JCA, stb.
- Webszreverek
    - Glassfish - referencia-implementáció
    - Oracle Weblogic
    - JBoss WildFly
    - Jetty
    - Tomcat, csak webkonténer

3. EJB

- **EJB:** Enterprise JavaBeans -
    - szabványos felülettel rendelkező, elosztott, szerver oldali komponensek, amik tartalmazzák az üzleti logikát.
    - EJB konténerben futnak, ami elfed minden féle milldeware sajátosságot
        - ezek: távoli eljáráshívás, szálkezelés, terheléskiegyenlítés, átlátszó hibakezelés, perzisztencei, tranzakciókezelés, életciklus, aszinkron üzenetek, biztonság
- **EJB Tipusok**
    - **1. Session Beean**
        - üzleti funkciók, függvényekként
    - 2. Enttiy Bean
        - Perzisztens tárolás volt, Objektum-relációs leképezés, JEE6-tól pruned -> helyette JPA
    - **3. Message-driven bean**
        - aszinkron üzenetkezeléshez
- **EJB verziók:**
    - **legfrissebb: EJB 3.2, JavaEE 7**
- **Explicit vs. implicit middleware:**
    - kép a diáról: elosztott objektumhoz csatlakoznak a szolgáltatás api-k, vagy közte egy interceptor
    - exxplicit: felduzzad a forráskód, hozzáragad az implementációt ki kell adni
    - **implicit middleware**
        - külön leíró fájl tartalmazza, milyen middleware szolgáltatásokat veszünk igénybe
        - Leíró fájl alapján **generálódik**
        - A forráskódban csak tiszta üzleti logika, a lleíró fájlt módosíthatja a vevő
- **EJB felépítése**
    - EJB 3-as Session bean
    - **Business interfész** - ha távolról is el akarjuk érni, különben csak a helyi webalkalmazásszerveren fogjuk elérni
    - **Csomagoló osztály** a helyi konténer által - megvalósítja a business interfészt

Implicit middleware az EJB 3-ban: egy EJB hívás folyamata



- **Referencia szerzése EJB példányra:**
  - ami hívja a metódusokat osztály , nem az implementációs osztályra hív, hanem **kliens oldali csonkora**
    - new-val nem lehet példányosítani
    - névszolgáltatás segítségével lehet megtalálni
    - hálózati kommunikáció miatt
- Interfészmentes nézet: már nem kötelező a business interfész
  - nem kell ha csak lokális kliens használja
  - nem lehet NEW-val, @EJB-vel vagy JNDI-vel keressük
  - a kliens osztály proxyt lát továbbra is

- **Névszolgáltatás**
  - Valamilyen néven regisztráljunk objektumokat, ezeket **binding** segítségével kötjük
  - hierarchikus: ejb/hu/aut/bme/MyEJB
  - **Kontextus: kötések egy halmaza**
  - **Direktory Szolgáltatás:**
    - névszolgáltatás kiegészítése, attribútumokat is lehet hozzátenné
      - DNS, RMI, NDS, sokféle van, LDAP-on keresztül egy részük elérhető Lighweight Directory Access Protocol
  - **JNDI** Java Naming And Directory Interface
    - egységes api
    - Telepítéskor minden EJB komponens, erőforrás kap egy JNDI nevet, amit a JDNI provider tart nyilván
    - a komponensek egymást JNDI név alapján keresik meg
    - pl.: bean neve: java:global/myapp/mybeans/BeanA
    - lehet közvetlen JNDI neveket , beregisztrálá alapján
    - Indirekt JNDI nevek
      - csak logikai névvel végezzük a JNDI keresést, egy leíró fájlban le van írva omponensre vonatkozik
- **EJB-hez szükséges dolgok**
  - Deklaránli kell, milyen middleware szolgáltatásokat akarunk
    - a.) Külön XML fájlban, neve: telepítésleíró **Deployment Descriptor - DD, felülírja az annotációkat**
    - b.) **annotációk**

- EJB csomagol: .class fájlok -> .jar (konténer által generáltak ofkórsz nincs benne)
  - META-INFejb-jar.xml a standard DD helye
  - több összetartozó EJB jar + Webalkalmazás .war-ja (Web archive) -> .ear (Enterprise Applicaion Archive)
  - jee6 óta egyből mehet war-ba minden. max 1db ejb-jar.xml
- telepíted az alkalamzásszerverre, ahogyan akarod
- függősége injektálása telepítésekkor kerülnek feloldásra

- **Session Bean**
  - EJB konténer garantálja, hogy egy implementációs osztálybeli példányt egyszerre egy szálból hív meg.
  - szinkronizációval tehát nem kell foglalkozni
  - de több konkurens klienst is ki lekk szolgálni
  - **ezért több példány is van** instance pool
  - **Állapotok**
    - **állapotmentes** - kliens nem számíthat arra, hogy végig ugyan azzal a példánnyal kommunikál
      - létezik <-> metódushívásra készen áll
    - **Állapottal rendelkező** - sessionhöz kötött, állapotát lementi, de nem nekünk kell. Passzívat vissza kell tölteni.
      - passív <-- -> (Ready <--> Bean Isntance does Not Exist
      - életciklust a konténer kezeli
      - annotációkkal lehet rájuk feliratkozni
    - **Singleton - egy példány lesz csak az EJB-ből**
      - klasztereknél minden könrnyezeten 1.
      - Alkalmazás telepítéskor létrehozható, nem szűnik meg rendszer szintű kivételkror sem
      - van köztük konkurálás, timeout egyéb megoldások

- **Annotációk**
  - Java **SE** újdonság 5 óta
  - Forráskódba illesztesz
  - Leírás.
  - Szintaxis:
    - @Annotáció(érték) ha csak egy van.
    - @AnnotációNeve(paraméter1=érték1, paraméter2=érték2…)
    - Tömböt is lehet
  - Néhány beépített SE 5
    - @Override: felülírok egy létező metódustt, elírások ellen
    - @Deprecated: elavult
    - @SurpressWarning: kussolj
  - Saját annotációk
    - @interface interfészben definiáljuk a nevet, és paramétereket
    - paraméterek metódus szintaxissal adhatóak meg, visszatérési típus lesz a paraméter típusa, metódus nevea paraméter neve
    - ha csak egy paraméter van, annak a “value()” név kell
    - lehet default értékük

```
...
@TODO(item="Javítani", severity=CRITICAL)
public void myMethod{...}
```

- A @TODO egy lehetséges definíciója:

```
package hu.bme.aait;
public @interface TODO{
   public enum Severity { CRITICAL, IMPORTANT, TRIVIAL,
   DOCUMENTATION };
   Severity severity() default Severity.IMPORTANT;
   String item();
}
```

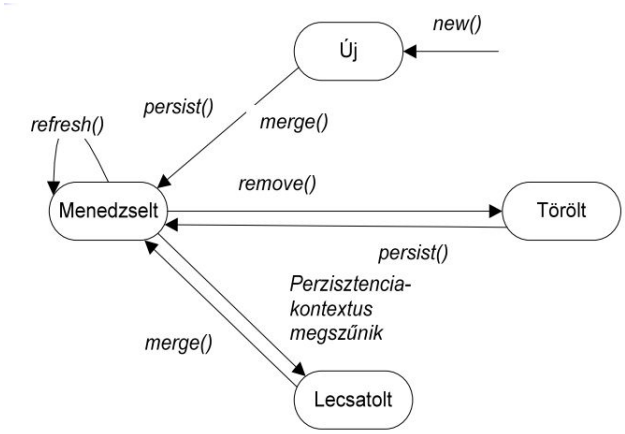
- Meta-annotációk
  - annotációk annotálására
  - @Target: megadható, mire alkalmazható az annotáció
  - @Retention: hogyan kezelje a compiler (ellenőrzi a compiler, belerakja a class fájlba, futás időben olvas, stb.)
  - @Documented: bekerüljön a javadoc-ba
  - @Inherited: öröklődik az annotáció?
- JavaEE annnotációk
  - ebből már nagyon sok van, az alkalmazásszerver-gyártóknak kell értelmezni
  - Telepítésleírók szerepét veszi át
  - Azért azok továbbra is használhatóak, de felülírják őket
  - **Session bean annotációk**
    - Távoli/lokális interfész / interfészmentes nézet
      - @Remoe, @Local, @LocalBean (intefészmentes)
    - Állapotkezelés definiálás
      - @Statful, @Stateless, @Singleton
    - Singletonok konfigurálása
    - Életciklus callback-ek konfigurálása
    - Függőség injektálás
    - Aszinkopr metódusok
    - Interceptorok
  - Singleton konfigurálás
    - @Startup - telepítéskor létrejön
    - @DependsOn függőség
    - @Konkurenciakezelés: @ConcurrencyManagement)
    - @Lock(WRITE) többszálú hozzférés
    - @AccessTimeout
  - Callback metódusok
    - a beanben nincsenek interfészek, helyette megjelölünk metódusokat
    - @PostContruct
    - @PostActive
    - @PrePassive
    - @PreDestroy
    - @Remove - elengedi az állapottal rendelkező session bean példányt
  - **Függőség injektálás**
    - @EJB ha ejb konténer, JNDI keresés
    - @Resource erőforrásokra
  - Aszinkron hívások

            - bármilyen tipusú session bean-re
            - vezérlés visszatér a klienshez, ielőtt a bean példányhoz eljut a kérés.
            - @Asynchronous annotáció
- **Interceptorok**
    - **Aspektusorientált programozásnál**
        - Teljes app-ra kitejredő szempontok, loggolás, bizotnsági ellenőrzés
        - valahogy őket összefogni
        - ezen **AOP** támogatás
        - megszakítják a bean metódusát, módosítják a paramétert, megakadályozzák a tényleges hívást, stb.
        - @Interceptors(MyInterceptor.class)
    - Interceptor írása
        - Java osztály, paraméter nélküli konstruktorra
        - @ArounnInvoke-kal van annotálva,
        - dobhat hibát, használhat függőség injektálást, EJB biztonsági és tranzakciós kontextusában fut
        - **CDI API** bevezeti a sima nem csak EJB-kre is.

4. JPA

- **ORM -** objektum-relációs leképezés, objektum-relációs mapping
    - entitás osztály <-> relációs tábla
    - entitás attribútumai <-> relációs tábla oszlopai
    - entitások -> relációs tábla sorai
- keresés: SELECT
- visszaírás: INSERT vagy UPDATE vagy DELETE
- régen az entity bean csinálta, mostmár nem.
    - képes volt automatikus O-R leképzésree
    - a memóriában lévő entitások és az adatbázis között a konténer szinkronizált
        - ezt amúgy a házim is így csinálja? nem, mert van enitás példány talán
- **JPA** JAva Persisntence API
    - entity bean nehézkes volt fejlesztői részről, teljesítményproblémák
    - helyette egy POJO Plain Old Java Object perzisztencia megoldást használunk
        - JPA entitások nem EJB-k
        - lehet SE-ben is használni
    - JPA 2.1 jelenleg EE7-hez
    - Api hívások során persistence providerrel kommunikál, ami lecserélhető
        - Toplink, Hibernate, Ebean
    - **perzisztens modulként viselkedik minden olyan jar, amenyek a META-INF könyvtárában van persistence.xml fájl.**
- Annotációk
    - @Entity - egy osztályra
        - implements Serializable kell
    - @Id - első kulcs,

        - @GeneratedValue strategy paramétere többféle generálós stratégia
    - getterek, setterek az eléréshez
        - @Access(FIELD) @Access(PROPERTY)
    - entitás attribútumok - tábla és oszlopnevek
        - @Table(name=”mytable”) vagy @SecondaryTable is, @Column(name=..)
- Beágyazott osztály
    - olyan osztály, ami nem él perzisztens entitásként, csak egy példányhoz kapcsoldódva
        - @Embeddable public class …. és benne attribútumok
    - felhasználni:
        - @Embedded >@AttributeOverrides({@AttributeOverride(name=”startDate”, column=@Column(“EMP_START”)), stb. és akkor be lehet illeszteni
- **Perzisztenciakontextus**
    - Perzisztencia provider által kezelt, memóriában lévő entitások halmaza **PC**
    - **EntityManager** interfészen keresztül érhető el, vele kezeljük az entitásokat, ő a kapcsolat az entitás és db között
    - Egy perzisztenciakontextus egy referencia.
    - PC-n belül minden entitás egy példányban
    - EntityManager életciklus kezelése
        - Függőség injektálás:
            - @Stateless a service-re általában
            - @PersistenceContext EntityManager em
        - Injektált EntityManager egy **menedzselt perzisztanciakontextust** fog elérni
            - EntityManagerFactory csak alkalmazás induláskor
            - EM a trazakciók eleje és vége között él
    - @PersistenceContext paraméterek
        - uitName: tbb unit is van a persistence.xml-ben, akkor meg kell adni melyik
        - tpye: élettartam
            - TRANSACTION - tranzakció végén haljon meg
            - EXTENDED - állapottal rendelkező sessiionbenhez kötött
    - Metódus típusok
        - 1. Entitás életciklusának kezelése
        - 2. adatbázis szinkronizáció
        - 3. entitások keresése
    - **Entitások állapotai**
        - **New**: csak memóriában létezik
        - **Managed:** létezik az adatbázisban, és hozzátartozik egy perzisztencia kontextushoz.
            - PC-n hívott flush() metódussal (EJB-ben automatikusan) beíródnak a módosítások
        - **Detached**: adatbázisban létezik, de nem tartozik a perzisztencia kontextushoz
            - nincs összekapcsolva
        - **Removed:** még létezik a perzisztencia kontextusban, de már ki van jelölve arra, hogy törölve lesz a DB-ből.



- ■
- ■ Új entitás menedzselté tétele:
  - persist() -> elsődleges kulcs ütközéskor exception
  - merge() -> elsődleges kulcs ütközéskor ráfrissít
    - visszatérési értéke a menedzselt entitás példány!
- ■ Entitás lecsatolása:
  - PC ürítéséve (em.clear()) vagy bezárásával .close(), entitás sorosításakor, vagy akár egyesével em.detach(entity)
- ■ Életciklus callbackek
  - Nem az EJB konténer hívja őket, hanem a perzisttencia provide
  - @prePersist, @PostPersist
  - @PreRemove @PostRemove
  - @PreUpdate @PostUpdate
  - @PostLoad

- ○ **Adatbázis szinkronizáció**
  - ■ EneityManager 2 metódusa
    - 1. flush() PC módosítások beírja
    - 2.refresh(entity) csak egy entitásra ráolvas DB-ből
  - ■ EJB-ben nem kell ezeket hívni, megteszi a konténer automatikusan
    - @EntityManager.setFlushMode()
- ○ **Lekérdezések**
  - ■ Mindegyik az EntityManager-en keresztül
  - ■ Keresés elsődleges kulcs alapján
    - <T> T find(Class<T> entityClass, Object primaryKey)
  - ■ a.)Lekérdezés teljesen dinamikusan
    - JPQL-ben: (EJB-QL)
    - public Query createQuery(String ejbsqlString)
      - select a from Account a where a.balance=:num1
  - ■ b.)**Statikusan** definiált, névvel azonosítható lekérdezés
    - public Query createNamedQuery (String anemOfQuery)
      - a lekérdezés a @NamedQueries-ben van deviniálva az entitás osztályban
  - ■ Query paraméterek:
    - setParameter
    - setMaxResuult
    - setFirstResult

                - getSingleResut
                - getResultList
                - executeUpdate
            - JPQL 2.1 -hez képest bővítés:
                - többes törlés, módosítás
                - JOIN, GROUP BY, HAVING, subquery
                - :paraméterNév a ?1 ?2 helyett
                - projekció Object[]
                - select-ben új objektumot csináljon
            - JPQL bővítés
                - Case, Nullif, Coalesce(első nem null-t veszi fel a listából)
                - index (listán belüli sorrendet ad vissza) Type, Key, Value Entry, In
- **Criteria API**
    - Deklaratív, string alapú JPQL objektumorientált, típusbiztos altenratívája a lekérdezéseknek
        - CriteriaQuery<Employee> cq = CriteriaBuilder.createQuery(Employee.class)
        - Root<Employee> emp = cq.from(Employee.class)
        - cq.select(emp);
        - cq.where(Criteriabuilder.equal) stb.
    - több kód, de típusbiztos
        - még van benne string navigáció (“emp.get(“lastName”)
            - Metamodel API segítségével kiküszöbölhető
            - metaadatok az entitásokról
            - ennotációk feldolgozásával tudja generálni a perzisztencia provider segédeszköze de megírható kézzel is
            - ezek a “Employee_” osztályok, Attribute map-eket tartalmaznak
            - Diesel query.where().eq(String.valueOf(Member_.username), username);
            - **kanonikus metamodel**
                - public static volatile attribútum, collecton-list-stb.
- Öröklés
    - JPA támogatja
    - **1. Egy tábla egy osztályhierarchiához**
        - **Discriminator** oszlop írja le a típust.
            - nem kell join!
            - nincs polimorfizmus
            - mély hierarchia esetén sok oszlop
            - nullázható mezők kellenek
        - legfelső szinten: @Inheritance(stragetgy=InheritanceType.SINGLE_TABLE)
        - @DiscrmininatorColumn(name=”oszlop”)
        - leszáramzottaknál: @DiscriminatorValue(“tupusra utaló név”)
    - **2. Külön tábla gyerekosztályonként**
        - Ősosztályban definiált oszlopok egy táblában
        - gyerekosztályban lévő oszlopok külön táblában + idegen kulcs az ősre
            - nincsenek fölösleges oszlopok
            - nem nullázható oszlop definiálható
            - polimorfizmust támogatja
            - sok join rontja a teljesítményt

        - @Inheritance(strategy=InheritanceType.JOINED)
    - **3. Egy tábla egy konkrét gyermekosztályhoz**
        - Külön tábla minden altípushoz
        - Tartalmazza az ősosztály attribútumait is
            - hatákony, de
            - polimorfizmust támogatni nehéz.
            - JPA nem követeli meg a támogatását
    - Egyéb megoldások
        - Entitás származhatn em entitásból
            - @MappedSuperClass őszotály, annak nem lesz külön táblája
        - Nem entitás származhat entitásból
        - Entitás **lehet absztrakt**
            - le lehet képezni táblákba, le lehet kérdezni
            - de ugye nem lehet belőle példány
- **entitások közötti relációk**
    - Kardinalitás szerint 4 típus
        - 1. @OneToOne
        - 2. @OneToMany
        - 3. @ManyToOne
        - 4. @ManyToMany
    - Irány szerint:
        - egyirányú
        - kétirányú (a kapcsolat mindkét végén lévő entitásnak lesz egy kapcsolatmenedzselő settere-gettere
            - **ezt a fejlesztő tartaj konzisztensen! !!!**
        - Kétirányú OneToMany = kétirnyú ManyToOne
        - Kapcsolatnak mindig egy tulajdonos oldala van
    - Példa:
        - Tulajdonos: Employee
          @ManyToOne
          @JoinColumn(name=”company_id”) //helyi mező neve
          @private Company company;
        - másik oldalon Company:
          @OneToMany(mappedBy=”company”) // a másikban
          private Collection<Employee> employees
        - meg getterek, setterek
        - lehet @JoinTable-t is használni, ha kapcsolótáblát használunk
        - **@ManyToOne kötelezően a tulajdonos** mert nincsen mappedBy paramétere, többi esetén tökmind1.
    - Cascade
        - mindegyikhez lehet cascade értéket beállítani
            - PERSIST
            - MERGE
            - REMOVE
            - REFRESH
            - ALL

            - fedault: nincsen cascade
    - Orphan removal
        - @OneToOne és @OneToMany kaphat orphanRemoval attribútumot, ez flusholáskor két esetben törli a kpacoslódó nem_szülő oldali aobjektumot
            - ha töröltük a szükőt (cascade=REMOVE-val egyenértékű)
            - menedzselt állapotban megszüntetnjük.
                - ha elávrult entitást más szülőhöz rendelünk, nem definiált a működés.
    - Fetch
        - mindegyik kapcsolatdefiniáló attnotációhoz megadható egy fetch @OneToMany(fetch=FetchType.LAZY)
        - azt adja eg, a betöltéskor betöltődnek -e a kapcsolódó entitások is
            - LAZY - csak ha rákérdezünk
            - EAGER - igen, töltse be
        - gondok:
            - lusta betöltés miatt még nincsen betöltve a kapcsolódó entitás, és ilyenkor lecsatoljuk - nem elérhetők lecsatolt állapotban a kapcsoloódó objektumok
            - megoldás: eager fetch, explicit gettert hív lecsatoláskor

5. Tranzakciókezelés

- célja: ACID
    - Atimicitás: vagy minden, vagy semmi
    - Konzisztencia: konzisztens állapotból csak konzisztens állapotba lépés
    - Izoláció: egymás állapotait ne lássák a folyamatok
    - Tartósság: harver hiba esetén is visszaállíthatóság
- konkurens adatelérés problémalehetőségei:
    - Dirty Read
    - Unrepeatable Read - egy lekérdezés később megismételve mást ad
    - Phantom readtd - kétszer ugyanaz a lekérdezés mást ad
- Izolációs szintek:
    - adatbázis tud segíteni, hogy melyik léphet fel ezen problémák közül.
    - Minél több problémát szűr ki, annál több zár kell, annál roszabb a teljesítmény
    - READ UNCOMMITTED: más által írt, de nem kommitált adatot tudok olvasni
    - READ COMMITTED: csak komittált adatot tudok olvasni
    - REPEATABLE READ: tranzakción belüli
    - SERIALIZABLE: konkurens tranzakciók sorosítva futnak

Izolációs szintek

Izolációs szint	Előfordulhat-e Dirty read?	Unrepeatable read?	Phantom read?
READ UNCOMITTED	Igen	Igen	Igen
READ COMITTED	Nem	Igen	Igen
REPEATABLE READ	Nem	Nem	Igen
SERIALIZABLE	Nem	Nem	Nem

- Zolációs szintek beállítása
  - csinálhatom én is
  - JPA használata esetén generált JDBC kód.
    - Az EJB specifikácó nem követeli meg, hogy lehessen definiálni ilyen szinteket -> alkalmazásszerver függő
- Elosztott tranzakciók
  - támogatni kell az alkalmazásszervernek
  - több tranzakciónális erőforrás, több szál vesz részt
  - tranzakciós kontextus ha egy szál csatlakozik egy tranzakcióhoz
  - minden bean példány ganartáltan egy szált ér el.
  - **EJB specifikáció szerint TILOS új szál indítása**
- Tranzakcióhatárok
  - generált wrapper csatlakozásánál
  - **Kliens réteg**
    - nem az alkalmazásszerveren futó komponens.
    - JEE nem követeli meg a támogatását
    - Hosszabb a tranzakció
    - kliens határán van a trantzakció határa
  - **Bean tranzakció**
    - forráskód tartalmazza a tranzakció indítást-lezárást-rollbacket
    - teljesen a programozó kezében van a döntés a tranzkicókezelésről
  - **Konténer általi dekleratív tranzakció**
    - konténer által generált wrapper tartalmazza a tranzakció indítását, visszagörgetését
    - lehet annotációkat teni @ApplicationExpceiton(rollback=true) szinten
- **Tranzakciós attribútumok**
  - Telepítésleíró, vagy annotációban leírhato, hogy dekleratívan kezeli a tranzakciókat.
- **Tranzakció session bean-ben**
  - Állapotmentes session bean -> nincs állapot -> nem kell visszaállítani
  - ha állapottal rendelkező -> SessionSycrhnization annotáció, és kell három függvényt megjelölni
  - Singleton session beannél konténer általt tarnzakciók,

Tranzakciós attribútumok az egyes EJB típusokra

Tranzakciós attribútum	Állapotmentes session bean	Session bean állapottal	Singleton session bean	Message -driven bean
Required	Igen	Igen	Igen	Igen
RequiresNew	Igen	Igen	Igen	Nem
Mandatory	Igen	Igen	Igen (kivéve @PostConstruct és @PreDestroy)	Nem
Supports	Igen	Nem	Igen (kivéve @PostConstruct és @PreDestroy)	Nem
NotSupported	Igen	Nem	Igen	Igen
Never	Igen	Nem	Igen (kivéve @PostConstruct és @PreDestroy)	Nem

- **JTA** Java Transaction Api
  - ezen keresztül történik a tranzakció kezelés
  - négy metódus
    - 1. begin()
      - tranzackió megkezdés,e létrehozása
    - 2. commit()
      - lezárás,
    - 3. rollback()
      - visszagörgetés
    - 4. setRollbackOnly()
      - vissza kell görteni már ezt a tranzakciót?
  - EJBContext-ből lehet elérni a tranzakciókat
    - meg azokat amiket csak a konténer tud
  - myCtx.getUserTransaction().begin()
  - Korlátok JTA-ra
    - EJB által menedzselt tranzakciónál nem lehet usertTranzakciókat
    - EJB által menedzselt tranzakcióknál nem lehet setRollbackOnly-t
- JAP és tranzakciók
  - ezek POJO-k, nem kaphatnak tranzakciós attribútumokat!
  - azért van rá workaround. EntytManager apin keresztül pl vagy persistence.xml-ben
  - JTA képes egyből a drivert megszólítani
  - Session Facade: webrétegbeli kliens -> session beanek -> tranzaciókezelés -> entityManager -> jpa entitások -> perzisztens műveletek.
  - műveletek
    - persist, merge, remove, refresh csak tranzakción belül
  - **Optimista konkurenciakezelés:**
    - Ha a @Version más, akkor optimisticLockException
    - persistencia provider kezeli
    - nincsenek zárak -> jobb teljesítmény
    - READ COMITTED izolációs szint
  - Explicit zárkezelés:

        - em.lock, izolációs szintek beállítása
        - csak tranzakicón belül hívható, verziószám növelések stb.

6. EJB lehetőségek

- **Biztonság**
    - Autentikáció: felhasználói azonosítás
    - Autorizáció: adott felhasználó jogosult a műveletek elvégzéséhez?
    - **JAAS**
        - Java Authentication adn Authorization Service
        - olyan modulok, amelyket hordozható
        - tipikusan web rétegben autentikálás, és a biztonsági kontextust felhasználja az EJB
        - Autorizáció
            - két mód,
                - a.) programozott biztonsági ellenőrzés
                - b.) deklaratív biztonsági ellenőrzés
            - A javaee **szerep alapú biztonsgi modelt használ.**
            - Telepítésleíróban abszolut szerepeket írunk le
            - EJB azt ellenőrzi, hogy a felhasználó megfelelő szerepben van-e?
            - Telepítésko meg kell adni, melyik felhasználó milyen szerepbe tartozik.
            - az autentikáció független az autorizációs szinttől
            - Programozott biztonsági ellenőrzés:
                - implementációs osztály kódja ellenőriz
                - EJBContext-ben vannak megfelelő metódusok
                    - myCtx.isCallerInRole(“admin”) stb.
            - Deklaratív ellenőrzés
                - telepítésleíróban, vagy annötációban @RolesAllowed
                - EJB metódusokra is
- **Timer Service**
    - Üzleti logika kívánhat időzítést
    - standard java-ban új szált indítunk, **de az jee-ben tilos**
        - mert pl. az EJB konténer nem tudná követni a tranzakciókat
    - Timer Service
        - A konténert kérjük meg, hogy indítson egy timer szálat
            - így tudni fog róla a konténer
            - EJBContainer hozza létre, de utána mi kommunikálunk vele
        - lejárhat, indítás, megállítás,
        - állapottal rendelkező session bean nem használhatja
        - Timer lejártakor meghívott callback metódus van @Timeout
        - LEhet őket bővíteni pl. időzítéses megoldásokkal
        - lTimer lehet perzisztens, vagy nem.
- **EJB Lite**
    - EJB egy részhalmaza
    - kisebb teljesítmény overhead, olcsóbb, egyszerűbb

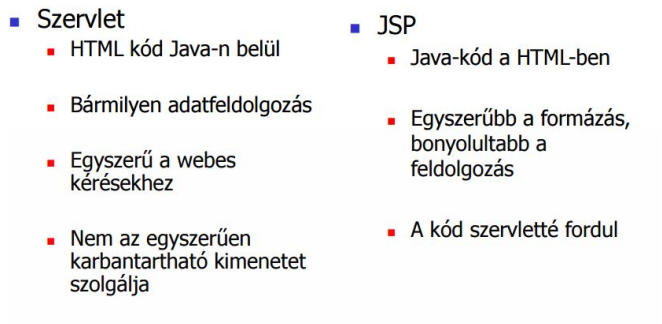
    - nincs benne:
        - Message-driven bean,
        - entity bean
        - távoli interfész
        - XML websozlgáltatások
        - timer service
        - Aszinkron hívás
    - JEE6 Web profile csak az EJB Lite-ot írja elő.
- **EJB 3.1 - Beágyazott EJB konténer**
    - Amikor nem karok egy egész JEE alkalmazásszervert futtatni, pl
        - unit teszteknél
        - egyszerűbb batch programoknál
        - dekstrop alkalmazásban
    - Ekkor JSE környezetben példányosítható EJB konténert hozunk létre, a classpath-ban lévő EJB-knek nyújt szolgáltatásokat.
    - Beágyazott EJB konténerek eddig is voltak, de mostmár szabányos api
        - EJBContainer sc = EJBContainer.createEJBContainer()
          Context ctx = ec.getContext()
          MyEJB myEjb = ctx.lookup(“java:global/MyEJB”);

7. Szervletek

- **Webalkalmazások**
    - szerver és felhasználó, TCP/IP-n kommunikálnak
    - Vékony kliens a király
        - nem kell feltelepíteni, csak böngészőből
        - statikus és dinamikus tartalmat kap eredményként: HTML, XHTML, XML stb stb.
    - Három réteg főleg MVC
        - Data Access Layer
        - Application Layer
        - Presentation layer
    - nincs éles határ!
- **Szervlet**
    - Java Thread alapú, objektum orientált technológia kérés-válasz protokollok szerver oldali kezelésére.
    - Skálázható, konténer szolgáltatásokat nyújt.
    - Java **objektum** amely má keretrendszerre és apira építve **bővíti más szerver funkcionalitását**
    - Az API a JavaEE spec részét képzi
    - hordozható komponens

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    public void doGet(
            HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String userName = request.getParameter("userName");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head>Sample servlet</head><body>");
        out.println("<h1>Hello" + userName + "</h1>");
        out.println("</body></html>");
        out.close();
    }
```

- kinézet:
- **JSP** Java Server Pages
  - elválasztja az alkalmazást a megjelenésről
  - Kiemnet: XML, XHTML, HTML
  - JavaBean-ek széles támogatása
  - Saját XML tagekkel, java osztályokkal bővíthető
  - Szervlet technológia, “kifordítva”
  - JSP oldal:
    - szöveges doksi
    - statikus és dinamikus részletekkel
    - .jsp, .jspf, .jspx



-
- Na most melyiket használjuk akkor?
  - Szervlet, ha
    - paraméter feldolgozás, új fájlformátumok, ábrák, (lol…)
  - JSP
    - html megjelenítés, formázás
  - átlalában a kettő együtt.
    - Szervlet: controller
    - JSP:view
    - vannak keretrendszere
- **JavaEE Webalkalmazás**
  - Definíció: egy telepíthető csaomag webkomponensek, statikus erőforrások, segédosztályok és telepítésleírókból
  - Megejenés: könyvtárak + fájlok VAGY .WAR (ami egy ZIP fájl) vagy .EAR-ként része az alkalmazásnak
  - maven
    - src/main/java: forráskódok

        - src/main/resources : mindenféle erőforrás fájl
        - src/main/webapp - jsp oldalak, statikus tartalom
        - src/test - tesztek
    - Speciális könyvtár: **WEB-INF**
        - nem publikus
        - konfigurációs fájlok, web.xml
        - JSP tag libary leírók
        - lefordított osztályok, /classes
        - .jar-ok
    - Servlet 2.5-ben nem kötelező a web.xml, ha csak JSP-t használunk
    - jelenleg: JavaEE 6
        - Servlet 3.0, JSP 2.2
    - következő: JavaEE 7
        - sevlet 3.1, JSP 2.3
    - telepítés
        - deploy, sokféleképp lehet
        - általában sima fájlmásolás
    - Context
        - alkalmazás környezete, konténer kezeli
    - **URL leképzés**
        - JAvaEE webkonténer HTTP szerverként is működik
            - HTTP kérések URL-jét dekódolja
            - Statikus tartalmat szolgál ki
            - Dunaimikus tartalmat a konfiguráció alapján szerlvetnek továbbít
        - Context root
            - alkalmazásra ellemtző egyedi elérési út
            - .WAR fájl neve
            - leghosszabban illeszkedő címre illesz a navigation
            - telepítés leíróban lehet mindenféle mintákat megadni, melyik útvonalra mi vonatkozzon
    - **Telepítésleíró**
        - **Standard: /WEB-INF/web.xml**
            - sorrend, kisbetű-nagybetű
        - **Szerverspecifikus: /WEB-INF/sun-web.xml**
        - tartalmuk:
            - életciklus vezérlők, kontextus paraméterek, szűrők, error mapping, biztonsági beállítások, session élettartam stb stb annotációk közül is lehet itt sok
            - <context-param>
                - név-érték práosok, globális beállítások, majd valamelyik komponensnek. name van meg <param-value>
                - servletContext.getInitParam(..)
            - <servlet> deviníciók, mappign, kiválható a @WebServlet annotációval
            - <filet> szűrők, @WebFilter
            - <error-page>-ek,
            - biztonsági beállítások
    - Összefoglalás:

        - .War fájlok
        - WEB-INF
        - web.xml
        - ezeket a megfelelő eszközök elfedik előttünk, de érdemes tudni mi mit generál.
- **Servlet Request-Response**
    - Szervletet általában HTTP-re használjuk, de akár FTP-re is implementálható
    - folyamat
        - 1. Fogadja a kliens kérését **Request**
            - 2. Kinyeri a megfeleő ifnormációkat **Parameters**
            - 3. alkalmazáslogika alapján valami generálódik **Parameters**
        - 4. visszaküldi a választ **Response**
    - Mi van a Reuqestben?
        - kliens adatai: ki, milyen adatok, fejléc, locale, stb.
    - **ServletRequest intefész: elérhetőek a**
        - Protokoll infók http, https-ről
        - adatfolyam tipusa
        - adatfolyam maga
        - kliens adatok
        - paraméterek
        - request scope objektumok
        - HTTP GET és HTTP POST paraméterek
    - Servlet response:
        - Szöveges és bináris, HTTP fejléc, cookie, cache
        - setHeader, setContetntType, addCookie, sendRedirect stb.
- **Szervlet környezeti változók**
    - **Scope object**
        - Infók megosztása a komponensek között, hashtábla jellegű,
        - nem ugyan az mint az oldalnak küldött paraméter!
    - **Web-context scope**
        - Egyetlen ServletContext objektum alkalmazásonként
        - Konfigurációs beállítások
    - **RequestDispatcher**
        - webes kérés átirányítása, szerver oldalon
    - **SessionScope**
        - csak kliens állapot karbantartása
        - HTTP -ben nincs állapot : cookie-k vagy parameter rerite kell.
        - HTTP Session tehát:
            - kérések közötti állapot megőrzésre kell
            - cookie-val, URL rewrie, Hidden form fields
            - Session timeout: 30 perc általában
        - a SESSION FONTOS ELŐFORRÁS
            - csak azt tegyük bele, amit igazán kell
            - Session objektum megosztott, több szálon is kezelhetik
- **Szervlet szűrők**
    - Java Servlet Filters
    - megfigyelni, módosítani, közbeavatkozni kérés és válasz alapján

    - szűrőket láncba szervezhet
    - Mit csinál egy szűrő?
        - Headert vizsgál
        - Request objektumot átalakítá
        - Kivételt dob, stb.
    - web.xml-ben lehet definiíálni a sorrendjüket
    - pl. LoginFilter
- **Szervlet életciklus**
    - GenericServlet
        - létrehozás - megsemmisítés
    - HttpServlet
        - doGet, doPost, doXXX
    - Kontténer hozza őket létre, fel lehet rájuk iratkozni
- **Többszálóság szervletekben**
    - Webkonténer több szálon fogadja a konkurens kéréseket
    - apalértelemzett esetben a webkonténer egy példányt hoz létre minden szerlvet oszályból, de a kliensek több szálon érik el őket
- **Servlet 3.0**
    - Webkomponens annotációk
        - web.xml nélkül is lehessen szerlvetet, filtert, listenert definiálni. annotációkkal.
    - web fragments
        - WEB-INF/web.xml-ben kereste a konténer eddig az információkat
        - mostmár modulárs a web.xml: WEB-INF/lib jar-ok META-INF könytárában is keres web-fragment.xml néven
        - ugyan az mint a web.xml.
    - Webkomponenske regisztrálása futási idően
        - keretrendszerk konfigurálásan kiküszöbölésére
        - alkalmazásszerver indulásakor a szerver services initializer-t keres
    - Aszinkron feldolgozás:

8. JSP

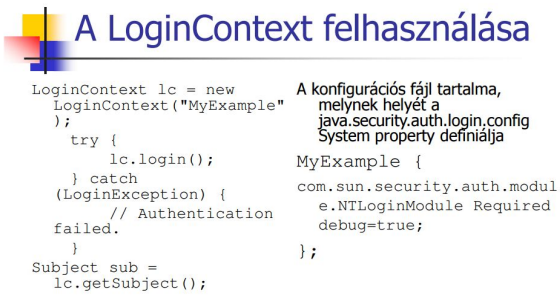
- Szöveges dokumentum, statikus és dinamikus elemekkel
- életciklus: ha történt változás újraforgatjuk
    - futtatás közben is, könyebb fejleszteni
- szintaxis
    - hagyományos
    - XML
- lehetnek benne szkript elemek
- szkriplet: oldalon generált változók, lokális változók, statikus tartalom
    - osztály, metódus definíciót nem tartalmazhat, csak implixit objektumokat max
- Akció elemek
    - jsp:elemnév
    - pl. feldolgozás továbbízása, más komponens beágyazása, pluginek, JavaBean osztályok
    - feflection

- **JavaBean != Enterprise JavaBen**
    - JavaBena= java objektum, amely mezői segítségével műveleteket foglal össze
- Tag libary:
    - JSP oldalak között megosztható, újrafelhasználató komponensek
    - Custom komponensek, saját akció elemek - tagekkel
    - <h: hello name=”TOM”/>
    - Java Standard Tag Libary JSTL
    - be kell az oldal elején deklarálni
- Saját tag
    - tag handler osztály
    - Tag Libary Desciptor
    - Tag file: tag naglder osztály alternatívája: segítségéel JSP-hez hasonlóbb a stílus
    - .tag kiterjesztés
- Tag file-ok használata
    - sablonok a gyakran ismétlődő felületekhez!
    - **tag file megkötés: scriplet nem lehet a törzsében**
- Tag Libary
    - tag-ek összeállított gyűjtemnye
    - átalában jar fájlként: MEAT-INF/taglib.tld
    - .*class fájlok
    - .tag fájlok
    - egyéb erőforrások
- Dinamikus attribútumok
- TagExtraInfo
- Környezeti változók és erőforrások elérése
- **JPS 2.0**
    - Kevesebb java kód JSP oldalon
    - fejlesztőbarát
    - **Scriplet**
        - Dinamikus tartalom megjelenítésére több lehetőség is rendelkezésre áll
        - **${kifejezés}**
- **Expression language**
    - **SPEL** (spring expression language)
    - JSP konténer felismeri, ${}
    - programozhatóan is elérhető
    - egyedi függvényívások
- **JSTL**
    - Java Standard Tag Libary
    - custom tag-ek gyűjteménye
    - **core**
        - prefix **“c”**
        - <c értékadás
        - elágazások
        - foreach
    - **XML**
        - prefix: “x”

        - XML doksik parsolása, kezelése
    - **SQL**
        - prefix: sql
        - eléggé necces használni, az üzleti logika kódban legyen!
    - **függvények**
        - prefix: fn
        - 16 üggvény, mint pl. hossz
        - String manipulációk pl.
- Hasznos JSP komponensek
    - HTML fromok építéséhez
    - Táblázatokhoz
    - Szerver oldalli fa komponesnekhez
    - stb… sok keretrendszer van már.
- **generált Szervlet**
    - **JPS végső soron szervletté fordul**
        - nincs meghatáozva a pontos módszer, implementáció függő
        - JSP engines
            - GNUJPS, Jakarta, Jasper
            - Sun IBM BEA Oracle

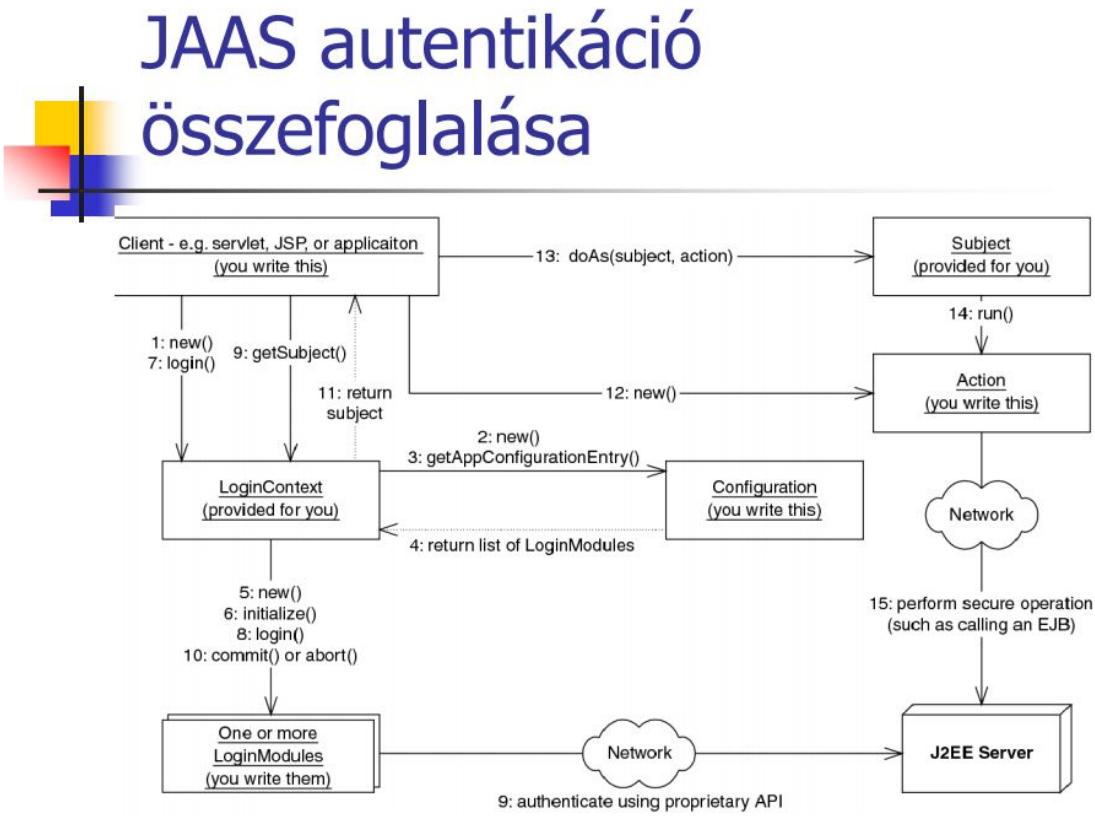
9. Java Authentication and Authorization Service

- **JAAS célja:**
    - Java 2 Security: felhasználó alkalmazásának megvédése a JVM-en futó alkalmazások elől
    - De a kliens-zerver alkalmazások fejlesztésénél fordítottt a cél-> felhasználók autentikálása
- **Autentikáció**
    - alkalmazásszerverek között hordozható autentikációs modulok
    - Ha egy JAAS modul autentikálja afelhasználót (a web rétegben) akkor a **megszerzett biztonsági kontextust az alkalmazásszerver továbbtejreszti az EJB-khez**
    - **LoginContext**
        - alkalmazás példányosítja
        - név hivatkozáss
        - sikeres login() után egy Subject objektum



        -
    - Principal interfész: entitás személyazonosságát reprezentálja
- **LoginModule**

- JavaSE5 tartalmaz beépített login modulokat, ezeket csak be kell konfigolni ,és már használható is
- Saját loginmodul fejlesztése a LoginModule interfész, és 5 metódusának implementálásával



-
- **Authentikáció javaEE-ben**
  - alkalmazásszervereknek **támogatniuk kell a szerep alapú autorzicáiót**
  - telepítésleíróban abasztrakt szerepek
  - telepítéskor kell emgadni mely csoportok/userek tartoznak az adott szerepekbe
  - Úgy írható meg az app, hogy nem tudjuk mi lesz az autentikációs mechanizmus
  - JAAS-re épül az implementációja
  - **realm-**ekben találhatóak a felhasználók és csoportok
  - pl. JDBC file és certificate realm.
  - írható saját is, de alkalmazásszerver specifikus
  - Serlvet 3.0 -> konténer által végzett bejelentkezés
  - programozottan vagy deklaratívan
- **autentikáció definiálása a web rétegben**
  - BASIC: standard böngésző ablak
  - DIGEST: jelszó titkosítva, de BASIC
  - FORM: általunk definiált login oldalt kell implementálni
    - vannak előírt adatai
- Szerepek webrétegben
  - Annotációval
    @DeclareRoles({“employee”, “manager})
    public class bármi…
  - vagy @RolesAllowed, @PermitAll

    - telepítésleíróval is
    - Programozottan:
        - getRemoteUser()
        - isUserInRole(String abstrractRole)

10 JSF

- Java Server Faces
- Háttér:
    - JavaEE => leggyakrabban webes appok
    - Servlet + JSP-vel minden megoldható, de nem a legkényelmesebben
    - Kezdő fejlesztnek szívás, jó lenn webes keretrendszer
    - Webalkalmazás keretrendszer legyen a JSF -> a **szabvány rész**
- Def
    - **Egy szerveroldali, komponens alapú felhasználó felület-keretrendszer, webes és általános környezetre.**
    - van rá giény ,egyszerű, nagy ipari támogatással
- Miért használjuk?
    - MVNC, UI koncepció webes környezetre
    - Komponens alapú
    - **finomabban hangolható, mint a JSP** felület elemeknek van állapota
    - JSP környezetben és anélkül is tud futni.
    - **JSF2.0-ban a JSP deprecated**
    - JavaEE 5-től része
    - ingyenes komponenskönyvtárak
    - újrafelhasználatóság
    - AJAX támogatás
- Architektúra
    - Böngésző/kliens -> JSF Controller -> JSP Page <-> backend
- JSF képességek
    - UI komponensek
    - Rendering model
    - eseménykezelés
    - validáció
- JSF verzió:
    - Jee7: JSF2.2
    - Jee6: JSF2.0
- JSF életcuklus
    - JSF oldal = UI komponensekből áll
        - 1. felépül a nézet
            - objektum hierarchia, bedrótozások
        - 2. request értékek beállítása
            - a komponens értékek beálltása, meg lettem-e nyomva, ki lettem-e jelölve
        - 3. validáció
            - validátorok hívása

        - 4. modell frissítésa
            - komponenseketfrissítuünk
        - 5. alkalamzás események hívása
        - 6. válasz rendeleés
    - Az életciklust a facesSerlvet bztosítja. egy URL mintára kell érkeznie aJSF-eskérésnek kb.
    - JSF-es kérésre JFS-es válasz, vagy nem JFS-es válasz is lehet
- Immediate=true
    - UIInputra
        - a 2. fázis végén validáció de a modell frissítése csak a 4. pontban
        - magasabb prioritású validációt kérünk, a 2. ponton validációs hiba van akkor a 3. állapot kihagyva azokra
    - UICommandra:
        - 2. fázis végén fut le az eseménykezelő,, a nem-immediate input validálódások és a modell frissítése előtt.
        - navigáció egyből megtörténhez (pl. Cancel gomb esetén nincs validálás)
- Komponensek renderelése
    - Render kit végzi
    - a komponensekhez tartozó tagekben lehet módostani hogy mie legyen
        - <h:commandButton> vagy <h:commandLink>
- JSF tag libarytk
    - html_basic
        - alap html
    - jsf_core
        - általános komponensek
- **Facelets**
    - JSP-JSF együttműködéssel votlak gondok
    - JSF kezedektől fogva jól pluginolható,
    - elterjedt egy JSF oldal definíciójáre agy technolgóia, **Facelets**
        - előnye: XHTML: bármilyen html editorral szerkeszthető
        - absztarkt szintaxisfává forudl
        - hibajelentések
        - validáció
        - template-ek
    - JSF 2.0 előírja a Facelets támogatását
    - JPS pedig deprecated ha JSF-et használunk -> új feature-ök csak Facelets-el mennek
    - Facelets tagek:
        - f: h: JSF komponensek
        - ui:
            - templatekezelés
            - komponensdefiniálás
            - debug infók
            - iteráció
        - c: JSTL core tagek egy része, if, when, choose, set, cache stb.
        - fn: JSTL függvények szintén
    - UI: composition vs u:decorate
        - mindkettő hivatkozhat egytemplate-re

        - composition-esetén a ui:define-okon kívüli elemek nem rendelődnek.
        - decorate-nál meg igen
    - Jellegzetességek
        - Kommenteket kiértékeli alapból
        - JSTL tagek használhatók, de core és fn-ne csak egy része
        - xml és sql tagek nem a nézeti feladat közé tartozik,...
- **ManagedBean**
    - Backing Bean = **Szerver oldali, a felhasználó felülettel összekapcsolt POJO objektum (=NEM EJB!) az MVC-ből a Model + Contoller logika testreszabásához.**
    - faces-config.xml-ben, vagy annotációval adhatóak meg.
        - @Managedbean(name=”myBean”)
    - állapotot reprezentáló property-ket, komponens példányokat tárolhatunk benne.
    - Tartozhat hozzá eseménykezelő
    - JSF oldalon Expression Language kifejezésként hivatkozunk rá:#{numberBean.value}
    - köthetjük a komponensbe írt értékeket managedbean porperty-khez
    - managedbeant első hivatkozáskor JSF implementáció pédlányosítja, ebeállítja, elvégzi az injektálásokat, scope-ba teszi
    - **managed bean mindig valamilyen scope-ban él** ez lehet request, session, applciation, none custom, vagy flash
        - @NoneScoped: minden hivatkozásnál létrejön
        - @ViewScoped: request és session közötti élettartam, AJAX
        - flash scope: egy redirect-nyivel tovább él mintha request lenne
- Navigálás
    - Deklaratív navigáció -> XML alapján
        - faces-config.xml
        - melyik oldal következik az action-ok kiementére, stringpárosok
    - Implicit navigáció -> kényelmes
        - JSF2.0
        - gyorsabb fejlesztés
    - Feltétles navigáció -> XML, annyira nem jó
    - Preemptív navigáció -> Get kérése,, bookmarkable
        - /entry.xhtml?id=#{blog.entryId}
    - JSF GET támogatás
        - régebben csak POST támogatás
        - nem voltak bookmarkolható url-ek
        - f:viewParam, amivel már lehet
- **Eseménykezelés**
    - **JSF1ben**
        - alkalamzás szintű események
        - ValueChanged event: valueChangeListener-re rákötni
        - PhaseEventek: JSF életciklushoz kötve
    - **JSF2ben**
        - SysrtemEventek -> hasonló a phaseEnventhez, nem alkalmazásspeciifkus, de picit több helyre lehet őket kötni
        - Lehet listenereket regisztrálni
            - programozottan

                - dekleratívan annotálva
- Konverterek,
    - vannak beépítettek
        - tipusok közötti, néhány közülük automatikusan
        - dátumra explicit ki kell rakni
    - saját konverterek: Converter interfészt valósítsa meg, getAsObject és gatAsString
        - render fázisban hívódik meg
    - annotációval vagy faces-config
- validátorok
    - négy mód:
        - 1. Beépített validátor, deklaratívan
            - nem sok van, rage validátorok főleg, + required
        - 2. alkalamzásszintű validáció
            - ez a FacesCoontext.getcurrentInstance.addMessage(New facesMessage (“shiiiit”);
            - tehát itt alkalmazás kódból szólunk ki
        - 3. Inline validáció
            - saját validátorok rákötése, meghívódnak
            - validateEmail(FacesContext context, UIComponent toValidate)
        - 4. saját validátor
            - meg kellvalósítani az interface-t , fel kell annotálni: írtál ilyeneket bőven
- Komponenskönyvtár ok
    - saját komponens írása nem túl egyszerű
    - inkább lopjunk
    - pl. MyFaces
    - vagy PrimeFaces…
- JSF 2.0 összetett komponensek
    - eleinte még nehéz volt saját komponenseket csinálni
        - kellett leszármaztatni a UIComponentből őket
        - összerendelés faces-config.xml-ben a megvalósító sztállyal
        - renderer megírása
        - renderer deklarásálsa stb.
    - 20-ban Java kód és XML konfigurálás nélkül fejleszthetőek
    - <composite:interface>
- Ajax
    - AJAX: Asynchronous JavaScript and XML
        - kliens oldali felület interaktívabbá tétele, miközben kommunikál kérések küldésen élkül a szerverre
    - JSF és AJAX:
        - jsf komponensek kiegészíthetőek kliensoldali JavaScript mgjelenítései elemekke, és szerver oldalon könnyű XML feldolgozással
        - JSF UI koncepció nagyon jól illik hozzá
        - HTTP kérés szerver oldali kiszolgálása történhet bármilyen technológiával,
            - válasz pedig egy JS callback-ben kezeljük, és úgy frissítjüka HTML DOM-ot
        - JSF-es custom tage-kt telejsen elrejtik a JS kódot
        - széles körűen lehet hekkelni

- Kliens viselkedések
    - f:ajax tag, validátorok és konverterek közös jellemzője:
        - tetszőleges komponenshez hozzáadhatók, és kibővítik a szülő viselkedését
        - ezek:
            - Kliens oldali validáció
            - DOM manipuláció
            - animációk, effektek
            - alert ablakok
            - billentyűzetkezelés
            - lazy fetch
            - kliens oldali loggolás
- Erőforrások
    - komponensek nagy része felhasznál külső erőforrásokat
    - CSS fáljok,
    - képek
    - JS fájlok
- tartalmazni kell a komponest tatalmazó oldalnak a hivatkzoást ezekre
- szervletek, fileterk rá hogy betöltsék
- mostmár 2 szabványos helyről is lehet
    - webalkalamazás resource mappája
    - META-INF/resources
- **JSF-JSP-JSTL interoperabilitás**
    - voltak problémák korán
    - Unified EL:
        - EL-ben a metódus átadása
        - EL-ben szintaxis bővítés
    - EL 2.2
        - JavaEE6: EL 2.2
        - nem JSF-specifikus, JSP-ben is használható Unified Expression Language

11 .XML webszolgáltatások

- XML nem:
    - jelölőnyelv, hanem szabvány
    - csak webes technológia
- **Extensible Markup Language**
- Jól formált, ha minden elem tag nyitótag után van zárótag, vagy csak zárótag
- attribútumok, időzeleje között
- egymás utáni elemek
- Sémák megadására lehet
    - DTD, XDR
    - XSD - XML Scghema Definition, maga a séma is XML
- vannak névtelrek, azonnos elemek megkülönböztetésére. h:table
- Feldolgozás
    - XSLT: Extensible Stylsheet Language Transformation

    - szabály alapú, eseményvezérelt programnyelv.
    - Document streaming:
        - nem kell az egész doksit, csak egy részét
        - parsol
    - DOM: Document Object Model
        - XML dokuemtmnum egészének beolvasás után áll elő
- **XML Api-k javaban**
    - **JAXP** Java API for XML parsing
        - interfész alapú
    - **StAX** Streaming Api for XML
        - Jee5 része, pull stream
    - **JAXB** JAva API for XML Biding
        - külön libaryként, JavaEE5 része
        - Kötések: Schema és mapped classes van összekötve.
        - **Schema -> document leírása**
        - **object -> JAXB mapped classes**
        - lehetőségek:
            - osztályok generálása
            - XSD is generálható java osztályból
            - marsahling, unmarshaling
- **XML Webszolgáltatás**
    - **olyan hálózati alkalmazás vagy komponens, amely SOAP protokoll és a WSDL nyelv fehasználásával XML dokumentumok formájában kommunikál a külvilággal.**
    - **jellemzők:**
        - XML mint adtprezentációs réteg -> platformfüggetlen!
        - SOAP: szállítási réteg
            - laza csatolás, önleíró
        - Lehet Szinkron, vagy aszinkron is
        - olyasmi, mint egy új elosztott objektum-orientált technológia
        - komplex dokuemntuom
        - **platformfüggetlenség miatt integrációs eszköz!**
    - Alapszabályok
        - meg akarunk hívni egy metósudt weben kersztül
        - válasz és a hívás is XML dokumentum
        - Kliens és a szerver megegyezik valamilyen üzenetformátumban
            - paraméterek? összetett infók?
            - ezekre lehet egyedi megoldásokat, de szabványos megoldásokkal lehet jól
- **SOAP**
    - **S**imple **O**bject **A**ccess **P**rotocoll
        - XML alapú kommunikációs protokoll, amivel le lehet írni
            - a metódushívást
            - a választ
            - a hibákat
    - **felépítése <soap:**
        - **Header**
            - gyakran üres, általában max middelware szolgáltatások

        - **Body**
            - tényleges metódushívás - visszatérési értkek
    - Protokollok:
        - **SOAP = XML + HTTP**
            - adatreprezentációt, hívás forgatókönyvet is definiál
            - Analóg a többi elosztott OO-technoógiával (CORBA, RMI)
            - Webszolgáltatásoknál is kell egy
        - **Interfész leíró nyelvre: WSDL**
- **WSDL**
    - **Web Services Description Language**
    - egy érvényes XML dokumentum
    - leírja a webszolgáltatás által adott műveleteket, metódusokat
    - minden összetett típust definiál
    - definiálha, milyen protokollokon és hogy hol lehet meghívni
        - emiatt önleíróak a webszolgáltatások
    - WSDL **szerződés,** minden információ tartalmaz
    - **tartalma:**
        - adattipusok <Type>
        - üzenetek <message>
        - prot típusok <postType> , tartalmazza az operation-öket
            - semmio köze a TCP potokhoz!
            - hivatkozhat a message-ekre
            - port típusok:
                - one way: csak <input>
                - notification: csak <output>
                - request resposne: kettő egymás után
                - solicit-respoinse fordított sorrendben a kettő egymás után
        - kötések
            - megadja, hogy az egyes űveletek milyen konkrét protokollokon érhetőek el - általában SOAP, de lehet HTTP vagy JMS is
                - <soap:binding> style attribútuma, rpc documetn
                - <soap:body> use attribútumra, encoded, literal
                - ezektől függ az üzenet strutkúrája
        - elérési pontok
            - <service> megadja az URLt amin elérhető a szolgáltatás
    - Szolgáltatások megrekesése:
        - **UDDI Universal Description Discovery and Integration**
            - egy directory szolgáltatás, amely webszolgáltotásokról tárol infókat
            - WSDL-t + egyéb dolgokat tárol
            - UDDI SOAP-ot használ
        - régen volt központi repo, de ez mára megszűnt, nagy cégek saját UDDI szerverei gyakoriak
        - van API-ja, futási időben is kereshet alkalamzás egy szolgáltatást

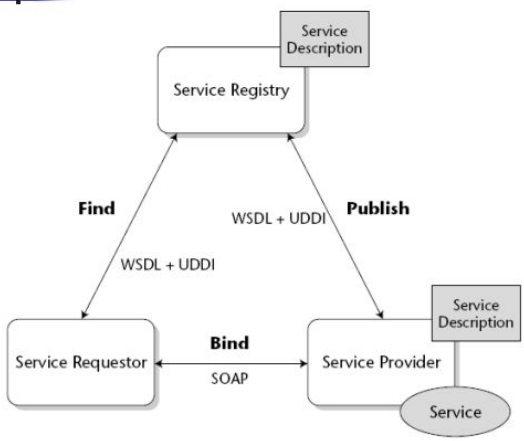
WS szabványok összefoglalása

UDDI	Telefonkönyv
WSDL	Szerződés
SOAP	Boríték
HTTP (SMTP, FTP)	Postás
TCP/IP (UDP)	Posta
Sémák (ma már csak XSD)	Szótár
XML	Ábécé

-
- **WSI**
  - szabványok egy része nem volt jól specifikálva -> nem mindegyik webszolgáltatás tud együtt működni
    - pedig erre találták ki
  - WS-Interoperability Organization, itt tisztázzák a nem egyértelmű kérdéseket

12 Webszolgáltatások 2

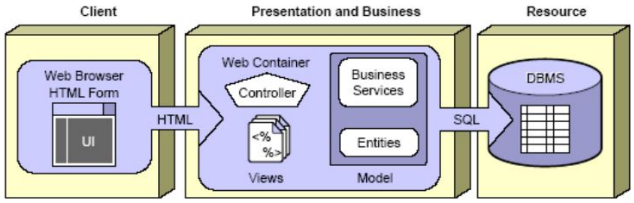
Kis emlékeztető



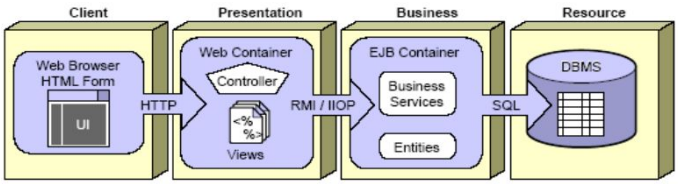
- WS = WSDL + SOAP + UDDI

-
- **Webszolgáltatások JavaEE-ben**
  - 1.4 előtt alkalmazásszerverek saját megoldásokat
  - 1.4 szabványosított
  - API-k vannak JAX-RPC, SAAJ, JAXR
  - JavaEE5 behozta az új api-t: **JAX-WS**

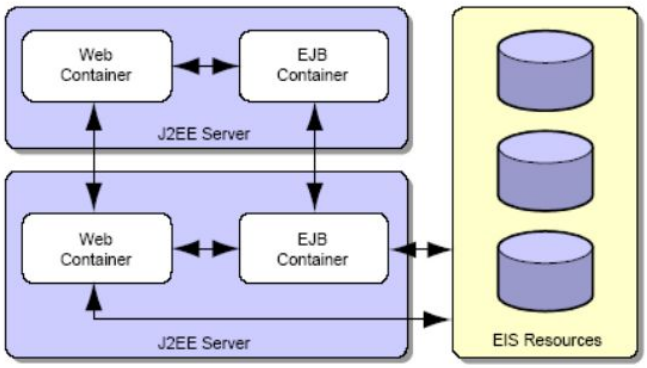
        - annotáció alapú
        - felhasználja a JAXB-t.
        - Két féle WS van:
            - **webkonténerben telepített**
            - **EJB konténerben telepített**
        - Elég egyetlen osztály megírása Service Implementation Bean
            - webrétegben ez POJO
            - EJB-ben stateless session bean
            - jelölés: @WebService
        - 2.2 óta minden metódust publikálunk, annotációval lehet kikapcsolni csak
        - WSDL részletei személyre szabhatókp
        - paraméterek és visszatérési értékek típusai a JAXB alapján képződnek el
        - webservices.xml telepítésleíró nem kell, elég egy file és működik is
        - **JAX-WS webszolgáltatás meghívása**
            - 3 lehetőség:
            - **1. Statius csonk**
                - WSDL fálj alapján ezeket generálja a JAX-WS eszköz, fejlesztési időben
                - kliens ggenerált Service gyermekosztályt egyszerű objektumként használja
            - **2. Dinamikus proxy**
                - Nioncs szükség a proxy előre generálására, az futási időben jön létre
                - interfészt meg kell írni
                - lassabb, de követi a WSDL bizonyos módosításait
            - **3. Dipatch interfész, üzenetorientált API**
                - se proxy, se interface
                - üzenetorientált API
                    - Stringként megadható a hívás törzse
                    - válaszban is teljes SOAP üzenetet kapunk
                    - kiváltja a XML->Java->XML tanszformációt
        - **JAX-WS lehetőségek**
            - implementáció lecserélhető
            - bővíthető
            - aszinkron hívás
    - **Web Services Invocation Framework WSFI**
        - Jakarta projekt, nem jee specifikáció része
        - osztálykönyvtár, ami segítségével szolgáltatásokat hívhatnunk protokollfüggetlenül
        - egyetlen követelmény: WSDL fájlban legyen definiálva a szolgáltatás
- **JEE alkalmazás architektúrák**
    - **1. Web-centrikus**
        - csak a webkonténert használjuk
        - ő tartalmazza az üzleti logikát, adatbázis elérést
        - mindenképp legyen azért ezen belül is MVC minta



- **2. EJB-centrikus architektúra**
  - EJB konténer tartalmazza az üzleti logikát
  - entities nem feltételnü lbean…
  - webréteg: controller + megjelenítés



- **3. B2B architektúra**
  - alkalmazásszerverek egyenrangú felek
  - Közvetlenül egymással kommunikáló EJB konténerek
  - XML üzenetekkel is lehet, akkor lazacsatolás



- **4. Webszolgáltatás alapú architektúra**
  - A webszolgáltatás telepíthető a web konténerben
  - Lazán csatolt
  - Több platfomról hívható
  - B2B és B2C együttműködésre is jó

- **BPEL**
  - Business Process Execution Language for Web Services
  - vezető technológia a WS-ek üzleti alaklamazásokban történő teljes körű felhasználást célzó technikák között
  - Ipari szabvány, nem java specifikus
  - egy XML alapú **szkritp-nyelv, amelyekben workflow-kat definiálunk**
  - **workflow:**
    - részben, vagy egészben automatizált üzleti folyamat, melynek során dokumentumok, információk és feladatok kerülnek átadásra és végrehajtásra a résztvevők között és által, procedurális szabályok alkalmazásával

- IDE-kben ez az XML szkirpt grafikusan is szerkeszthető
- BPEL-kompatibilis engine eket értelemio, és tudjha az ott definiált szolgáltatásokat nyújtani
  - pl. Acitivi
- BPEL nyelvi elemei
  - várakozás eseményre
  - üzenet transzformálása
  - elágazás
  - külső szolgáltatás hívása szinkron-aszinkron módon
  - hibajelzés küldése-fogadása
- BPEL és JAVAEE
  - több alkalamzásgyártó nyújt ilyet
  - BPEL konténer

13. JAX-RS

- **Java API for RESTful Web Services**
- Jee6 óta: JEE7-bern JAX-RS 2.0
- **célja: támogatás a REST stílusú webszolgáltatások fejlesztésére**
- **Representational State Transfer:**
  - olyan **szofvterarchitektúra,** amelyben különféle **erőforrások URI alapon érhetőek el.**
  - kliens és szerver között olyan dokumentumok utaznak, melyek erőforrások állaptoait repreztentálják
  - XML, HTML, JSON, kép, szöveg stb.
- **RESTful webszolgáltatás**
  - HTTP fölött URI alapon, egyszeűren címezhető szolgáltatás, amihez kel:
    - 1. szolgáltatás URI-je
    - 2. szolgáltatás által támogatott MIME type
    - 3. Szolgáltatás által támogatott HTTP metódusokat (GET, POST, PUT, DELETE)
  - nincs rá szabvány olyan, mint a SOAP vagy WSDL-nél
- tehát egy link, rákattintasz, és a GET, POST DELETE stb. parancsokkal már távol is végzel műveleteket rajta.
- **JAX-RS**
  - RESTful webszolgáltatások JAVA-ban lehetne egyszerű Servlettel is, de a JAX-RS magasabb támogatást ad
  - jellemzők:
    - POJO alapú
    - HTTP
    - sok formátum, bővíthető

JAX-RS példa

```
@Path ("books")
@Produces (MediaType.APPLICATION_XML)
public class BookResource {
   @Context
   UriInfo uriInfo;

   @GET List<Book> listItems() { return getAllBooks(); }

   @POST
   @Consumes (MediaType.APPLICATION_XML)
   public Response create(Book book)
        throws ItemCreationException {
        Book newBook = createBook(book);
        URI newBookURI = uriInfo.getRequestUriBuilder()
                .path(newBook.getId()).build();
        return Response.created(newBookURI).build();
   }
...
}
```

- erőforrások:
  - POJO-k
- futhat az alkalmazás SE, Servlet, vagy JavaEE környezetben is
  - **állaptomentes EJB is lehet JAX-RS erőforrás**
- Erőforrások életciklusa
  - minden kéréshez új példány ön létre
  - kosntruktor bemenő paraméteri injektálhatóak: Context, HeaderPara, cookieParam, stb.
  - Ha több konstrukt is van, a legtöbb aparméterű hívódik meg
  - tagváltozókat is lehet injektálni
- Resource metódusok
  - publikus metódus, annotációval
  - bemenő paraméterek injektálhatóak annotációval
  - visszatérési értékek void, Response, genericEntity vagy más
- **URI sablonok**
  - a @PAth bemenő paraméterében adható meg, az URI egyes részeinek parszolására használható, salbonban lévő névre hivatkozul
- Média típusok:
  - kérések és válawszol formátumát annoptációval definiáljuk
  - Resource
- Providerek:
  - segítségükkel bővíthető a JAX-RS implementáció
  - egy pldányban jönnek létre, injektálhatóak a @Context-tel a konstruktorparaméterek
  - tipusok:
    - entity provider: deszerializáció
    - context provider: injektálás
    - exception mapping proider: kivélettípusokhoz
- Kontextuso:
  - @Context-tel injektálhatóak resource osztályokba, proivederbe, vagy Application gyerekosztályokba
  - Context providerrel bővíthető az injektálható kontextusok köre
  - támogatott ajták:

        - Jeebeanek, CDI, Request, Providers blabal

14 Bean Validation

- JEE6 hozza be
- **Cél: ne keljen ugyan azt a validációs logikát több helyen, pl. webrétegben, perzisztenciarétegben is megvalósítani**
- Nem kötődik más technológiához
- egyszerű JavaBean-ekhet fűz validációs metaadatokat
- annotáció Vagy XML-lel felüldefiniálható
- felhasználható JavaSE, EE, JPA és JSF környezetben is
    - pl: public class Address {
    - @NotNull @Size(max=30)
    - private String addressline1;
    - @Size(max=30)
    - private String addressline2;
    -
- **Validációs kényerek**
    - azaz **Constraintek**
    - annotációk, paraméterekkel, melyek validációs követelményeket írnak elő tagváltozóra, metódusokra, típusokra, stb.
    - beépítettek:
        - @AssertFalse, @AssertTrue
        - @max, min, decimalmax
        - Digits
        - Future, past
        - Null, NotNull
        - Size
        - Pattern
    - lehet sajátot is írni, akár hozzákötve más annotációkat
    - validációs folyamat
        - osztályokon, interfészen a példányok teljes állapotára
        - változón: a változót elérve validál
        - metódus: csak getterre, annak az eredményét ellenőrzi
        - JPA entitás osztály konzisztensen a JPA annotációkkal
        - teljes objektumgráfok is validáhlatóak rekuirzívan
        - constraintek statikus metóduson, statikus változón nem elhelyezhetők. (minek?)
        - validációs osztály indítása:
            - Validation.build…...getValidator().valudate stb.
        - **Üzenet interpoláció**
            - **Célja, hogy a végleges validációs hibaüzenetet előálltísa**
            - Constraint-ek felülírhatóak, van message property maibe beleírhatunk
            - lokalizált változat
    - **JPA 2.0 integráció**
        - ugye rájuk is lehet tenni, de mikor hívódnak meg?

        - integráció szintje a persisntece.xml validation-mode elemében állítható, értékei:
            - **AUTO** ez a defualt.
                - pre-presist, pre-update, pre-remove fázisokban
                - csak akkor fut le, ha megíruk a Bean Validation-t
            - **NONE:**
                - a prezisztenciaprovider nem futtat validációt
            - **CALLBACK:** olyan, mint az auto, de kivétel dobódik ha nincs Bean Validation implementáció
    - **JSF 2.0 integráció**
        - Bean Validation implementáció jelenléte esetén a JSF validációs fázisában automatikusan meghívódik olyan opjbektumokra, amelyekhez **imput** értékeket kötünk.
            - letitlható, szelektálható…

15 - CDI

- **Context and Dependecy Injection for the Java EE platform**
- JavaEE6 vezeti be
- **alapfogalmak**
    - Függőséginjektálási lehetőségm a hava5-ben nem volt elég mindenre
        - egyszerű java osztályba nem injektálható EJB, DataSoiurce, EntityManager stb…
        - egyszerűj java osztály sem injektálható, csak EJB
        - ezekre DI keretrendszereket használtak, **Spring például**
    - **CDI célja:**
        - **függőség és kontextus injektálás kiterjesztése POJO-kra**
        - **EJB-k közvetlenül használhatók legyenek JSF managed bean-ként**
        - reuqest, session, application scope-ban tárolt objektumok életciklusának deklaratív menedzselése
    - **@Inject:** injektálható tagváltozó, kosntruktor vagy metódus megjejlölésére
        - mepgróbálja feloldani a függőséget
        - kosntruktok vagy metódus esetén a beenő paraméterek oldja fel
- ha nincs CDI: akkor Factory vagy ServiceLocator alkalmazás DefaultTimeSource.getInstance() és hasonlók
- ha több jelölt is van, @Qualifier annotációval definiáhatunk választást
- **Annotáció tipusok**
    - **@Named(“név”)** ezzel adható meg a neve mivel elérjük, camelCase default
    - Alapvetően egyszer használatos eszköz. Ha valamikor fel akajuk használni, akkor @Interface-ként definiáljuk és **@Scope** annotációval jellöljük
    - **@Sinleton:** csak egyszer
- **Új managed bean fogalom és a CDI**
    - Managed Bean: Java objektum JavaEE környezetben, szolgáltatásokat és életciklust kap a kontéenrtől
    - @ManagedBean
    - CDI, interceptorok, életciklus, stb. JNDI-ben regisztrálva
    - JavaEE6-tól pl. managed bean a **JSF managed Bean, a Session EJB,** így **elmosódik a határ a web és EJB között**

- Lehet típust is szűrni: @Typed(Shop.class)
- ha nem sikerül valamit feloldani: injectedDao.isUnstatisfied()
- **Beépített qualifierek**
    - **@Named**
    - **@Any** mindegyik beanen rajta van, akkor is ha nem írod oda
    - **@Default** alapból
    - **@New**
- **Scope-ok**
    - **@Dependent:**
        - alapértelmezett scope, injektálás céljához kötött az élete
    - **@Application Scoped, @RequestScoped, @SessionScoped** “ahogys a szervleteknél”
    - **@ConverstaionScoped:** session egy része
    - sajtá scope-ok
- Producer metódusok:
    - injektornak gyűrthatunk vele injektálandó objektum példányokat, ergo “magával húzza”
- Disposer metódus
    - takaríthat az injektált példány után
- **Hogyan injektáljunk?**
    - **A korábbi @EJB, @Resource, @PersistenceContext, @PersisnteceUnit, @WebServiceRef helyett használhatunk @Inject-et saját qualifierrel**
    - @Qualifier @Retention(RUNTIME) @Target({TYPE}) public @interface CustomerDatabase{} @Produces @Resource(lookup="java:global/env/jdbc/CustomerData s ource") @CustomerDatabase DataSource customerDatabase; @Inject @CustomerDatabase DataSource customerData;
    - De csak Dependent scope lehet Resource-oknál
- **Alternatíva**
    - OIlyan osztály, amit csak akkor kezel a CDI, ha explicitt kérjük a WEB-INF vagy META-INF könyvtárban elhelyezett beans.xml-ben
- **@Stereotype**
    - Együtt gyakran előforduló Scope-ok, interceptorok összefogására
- **Interceptorokat** is lehet kötni CDI annotációval
    - @Audited, @Interceptor, @InterceptorBindign stb.
    - explicit kell őket engedélyezni a beasn.xml-ben, sorrendjüket is itt írjuk le
    - EJB intercveptor a @Interceptors hamarabb hívódik meg, mint a CDI interceptor
- **Dekotárotork**
    - Interceptorokhoz hasonló konstrukció
    - olyan managed bean, ami implementál egy típust, és ezen típis többi megvalósításának (azaz a dekorált típusoknak) metódushívásait megszakítja.
    - Dekorátorokat is explicite kell definiálni
    - interceptorok után hívódnak meg, pl. pénzemek átváltása
- **Eventek**
    - hívók és hívási célok laza csatolását oldja meg
    - @Injext Event<fraud> pl.
    - fraudEvent.fire(frauld)
    - fel lehet rá iratkozni
    - szűrhetőek

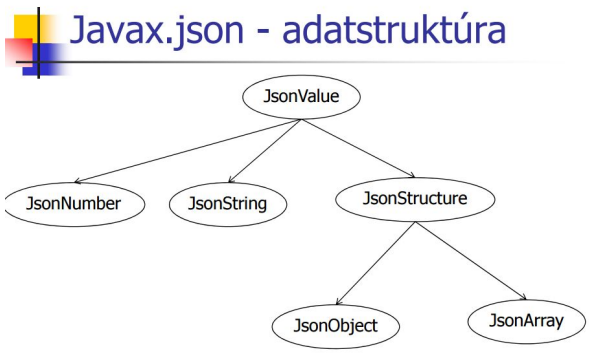
- observer metódus sok lehet, akár beanen is
- interceptor és dekorátor nem tartalmazhat observer metódust
- tranzakciónál megmodnható, hogy mikor sütődjön el
- meghívási sorrendjük **nem garantált**

16 JavaEE 7 újdonságok

- **fő célok:**
  - **HTML5 támogatás**
    - dinamiokus, interaktív alkalmazások létrehozása
    - WebSocket technológia bevezetése a válaszidők csökkentésére
    - JSON támogatás adatok cseréjéhez
    - továbbfejlesztett JAX-RS a skálázhatóság, aszinkrn működés és nagy teljesítmény biztosításához
  - **Egyszerűbb fejlesztés**
    - jogg kódhatékonyság, kevesebb boilerplate
      - default DataSource, default JMS ConnectionFactory, előre definiált JNDI nevek
    - Annotációk széleskörűsége, POJO-k előtérbe kerülnek
    - Kliens oldali API RESTful webszolgáltatásokhoz
    - Vállalati igények
      - többszűlú konkurens feladatok kezelése a skálázhatósághoz
      - kötegelt fealdatok érszekre bontása
- **Új technológiák**
  - **Java API for WebSocket**
    - **Websocket:**
      - Kétirányú kommunikáció létrehozása a szerver és kliens között egyetlen **TCP socket** segítségével
      - Kliens-szerver közötti valós idejű kommunikáláshoz
      - HTML5 része
      - részei:
        - Handshake protokol, válaszok
        - Adatcsere
    - Egyszerű használhatóság
    - Események érzékelése, kliens fel-le kapcsolódása pl.
    - szöveges és bináris üzenetek támogatása
    - handshake protokoll is felüldefiniálható
    - Végpontok összekötése: annotációval, vagy programozottan
      - **Endpoint** minden kacpsolthaoz
      - **Session:** végpont és a peer közötti interkaciók sorozata, egyedi infókat is lehet tárolni benne
    - Kliens oldali API-t is tartalmaz, pl. vastagklienshez
    - **Decoder/Encoder**
      - Java primitív tipusokra beépített
      - streamelés, szöveg vagy bibnáris olvasás,

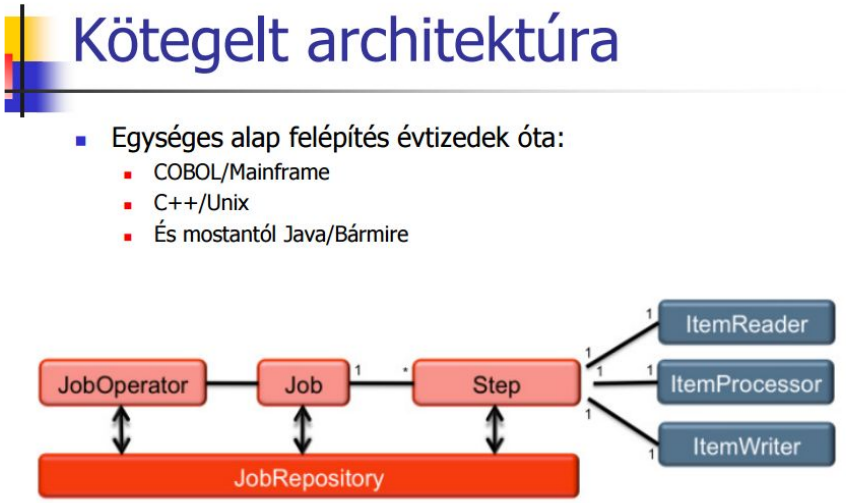
- **Kommunikáció**
  - **Üzenetfogadás**
    - MessageHandler segítségével, sessiönönként egy regisztrálva
  - **Üzenetküldés**
    - Session segítségével
- **Annotációk**
  - osztályszintű annotációk
  - metódusszintű annotációk pl kapcsolat létrejötte, lezárása
  - üzenetküldéskor meghívandó annotációk
  - hiba esetén meghívandó dolgok
- Beálltíás
  - ki a szerver, ki a kliens? a beállítástól függ

- **Java API for JSON Processing**
  - **JSON**
    - **JavaScript Object Notation**
      - kis méretű, szöveg alapú szabvány
      - ember által olvasható formátum adatcserére
      - kulcs-érték páropsok
      - igen, JavaScript nativ objektumleíró része
      - RESTful webszolgáltatások gyarkan ezt használják
      - Objektumok, tömbök, adattípusok
      - JSON a HTTP csomag törzse, a fejlécben Content-Type: application/json
    - **Json adatok kezelése**
      - **a.) Object model**
        - fa reprezentálja az adatokat
        - navigálható, analizálható, gyors
        - kimenetet egyszerre generálja
      - **b.) Streaming model**
        - Esemény alapéú, e3gyszerre JSON adat elemeket olvasunk
        - csak ha az adat egy részére vagyunk kíváncsiak, jó nagyon
  - **Java API for JSON Processing**
    - eddig 3rd parti megoldások voltak, mostmár része a jee-nek
    - osztályok
      - **Json**
        - parser-builder-generator statikus pléldányosító metódusok
      - **JsonObjectBuilder JsonArrayBuilder**
        - objektum vagy tömb modell létrehozáa laklmazás kódból
      - **JsonReader**
        - bemeneti JSON streamből objektum mdell
      - **JsonWriter**
        - stream-re ír egy objektum modellt



-
    - **Stream**
        - esemény alapú feldolgozó JSON adatok olvasására
    - **JsonGenerator**
        - JSON adatokat írhatunk vele egy streamre, elemenként
- **JSON és JAX-RS**
    - JAXB-hez hasonló binding nincs
    - de van egy jersey implementáció

- **Batch Applications for the Java Platform**
    - **Kötegelt eljárások (Batch jobs)**
        - olyan feladatok, melyek a felhasználó közbeavatkozása nélkül elvégezhetőek
        - Nagyvállalalti rendszerkben szokták használni
        - Nagy mennyiségű informcáióval dolgozó feladatok, háttérbenfuttathatóak
        - peridoikusan
        - futhanakpárhuzaomsan, akár ütemezhetőek
        - pl: számlázás, jelentés, adatkonverzió, képfeldolgozás, kockázatértékelés, készletgazdálkodás, portfólió optimalizálás stb stb.
    - **Batch Applications for the Java Platform**
        - **kötelegelt eljárások definiálsáának, implementálásának és futtatásának támogatása**
        - **a fejlesztőknek csak a kötegelt alklamazás üzleti logikájára kelljen koncentrálni**
        - Eljárások megadása XML alapú
        - Annotációk, interfészek üzleti logikához
        - **Batch konténer**
        - Képes ütemezni, és jól kihasználni az erőforrásokat
        - **jobs, steps, repositories, reader-processor-writter minták**



- **Kötegelt architektúra**
  - **job:**
    - Egy entitás, ami megába foglalja az egész kötegelt folyamatot
    - konténer a logikailag összefüggő lépések számára
  - **Step**
    - lépések összessége és azok végrehajtási sorrendje
    - ezek sorrendhelyesen határoznak meg egy feladaot
    - minden infót tartalmaz az aktuális feladat elvégzéséhez
    - lehet egyszerű, vagy összetett
  - **Blokkorientált lépés**
    - egy összetett STEP
    - több elem blokkosított beolvasásaChunk-orineted step
    - Adatot dolgoznak fel három lépésben
      - 1. egy valami beolvasása
      - 2. végrehajt egy műveletet
      - 3. letárolja az eredményt
    - hosszú futás, nagy adamennyiség
  - **Feladatorientált lépés**
    - feladatokat hajtanak vére, mint pl. könyvtár létrehozása, fájl mozgatás stb.
    - gyorsabbak
  - **Flow, Split**
    - ergy folyamon (**flow)** belüli lépések sorosan hajtódnak végre
    - folyamatban klévő lépések egy egységet alkotnak
    - Több folyam párhuzamosan is végrehajtható, ha **split-** be teeszük
  - **Állapot és döntési elemek**
    - állapotokat nyomon követ a keretrendszer
      - pl. jelez, ha véget érte
    - döntési feladat,
      - meghatározza a következő lépést
  - **ellenőrzőpontok kezelése**
    - fontos elvárás, hogy a hosszan futó alkalmazások rendelkezzenek ellenőrzőpontokkal, és lehessen őket onnan folytathatóak

            - egy lépés periodikusan lementheti a jelenlegi állapotát, így az utolsó konzisztens állapotból újraindítható
            - “save pont”
            - zárolásokkal jár ilyenek létrehozása, szóval sakkozni kell a gyakroisággal
            - **egy lépés végén mindig van ellenőrzőpont**
    - **Kövewtelmények a keretrendszerrel szemben:**
        - feladatok, lépések, döntési elemek és kapcsolatok meghatározása
        - lépés csoportok futtatása
        - feladatok, lépések állapotkezelése
        - futtatni a félbeszakított feladatokat
        - kezelni a hibákat.
    - **Batch alkalmazás felépítése**
        - XML fájlok: **Job Specification Language**
            - feladat és lépéseik
        - Java osztályok
            - üzleti logikák
    - **Job Specification Language**
        - **A job szerkezetét adja meg**
        - @Named annotációval ellátott Java osztályokra hivatkozhat
        - megadhatók tulajdonságok, paraméterek, listeneker
        - akármenyi JSL fájlt létrehozhatunk, META-INF/batch-jobs
        - Batch runtime ezeken a lépéseken megy végig sorban, és akkor ér véget, ha
            - a.) véget ér, mihnd sikeres
            - b.) valamelyik elbukik
    - **Folyamat indítása és állaptoának lekérdezése**
        - feladatot tartalmazó fájl nevét adjuk meg a jobOperator.start(---)nak
        - Joboknak adatokat prefixekben átadhatuznk
        - Állapotát ki tudjuk olvasni
    - **Futás lépései**
        - 1. Tranzakció indítása
        - 2. ItemREader hívása és az általa olvasott elem átadása az ItemProcessornak
            - ő feldolgozza a kapott elemet, és az eredményt
            - visszadja a batch runtime-nak
        - 3. A batch runtime megismétli item-countszor a 2-es lépést, miközben számon tartja a feldolgozott elemek listáját
        - 4. a batch runtime meghívja az ItemWritert, ami item-count darabnyi feldolgozott elemet kiírá
        - 5. Ha hibát dobvalamelyik ItemXX, akkor a tranzakció sikertelen, és FAILED állapot
        - 6. Ha nincs hiba, a runtime megszerzi az ellenőrzőpontokat, és commitolja a tranzakicót.
        - ismétlés
- **Concurrency Utilities for Java EE**
    - **Konkurencia kezelés**
        - Két- vagy több feladat végrehajhtása azonos időben
        - Két fő eleme:
            - **1. folyamat:** maga az alkalamzás, saját erőforrásokkal, pl. memórával
            - **2. szál:** osztoznak a folyamattal az erőforrásokon éa a futtatási környezeten

            - könyebb létrehozni és kevesebb erőforrást foglal
        - Mamár majdnem minden gépben több mag és több processzor is van
        - pár magos CPU több ezer szálat tud kezelni
        - sok problémát megold a konkuenrs futás, de sokat fel is vet
            - éhezés
            - deadlock
            - inkonzinsztencia
    - Az API-nak a **célja, hogy növelje az aszinkron képességeit a komponensnek** az alábbiak segítségével:
        - Felügyelt végrehajtás
        - Felügyelt ütemezés
        - Felügyelt szál létrehozás
        - Megfelelő keret szolgáltatás
    - ezt korábban nem tudta a JavaEE. **EJB-ből tilos szálat indítani…**
    - JAvaSE5 vezette be a concurrent csomagot, amely magas szinten támogatja a konkurens működést, ezt használja fel az ap**i.**
    - **Biztosított elemek**
        - **MAnagedExecutorService**
            - aszinkron feladatokhoz, konténer felügyeli
            - Future objektummal érhető el az eredmény
        - **ManagedScheduledExecutorService**
            - Ütemezés megvalósítása, hogy adott időben fussanak le a feladatok
            - Késleltetés, periódikus hívás
        - **ContextService**
            - dinamikus porxy-k, hogy a konténer kontextusát használhassa az alkalmazás
        - **ManagedThreadFactory**
            - konténer által kezelhető szálak létrehozása, beállítása
    - **Notifications**
        - lehet listenekereketfeliratkoztatni
        - értesítést kapunk a hibákról, futás kezdéséről, befejezőedésről, stb.
        - naplózásra tökéletes
    - **Tranzakciók?**
        - comit és rollback nehézségek
        - JTA -t használja a JavaEe, hogy biztosítsa ezek futását biztosan.

17 JavaEE7 újdonságok

- **EJB 3.2**
    - **Message Driven Listener marker**
    - **Stateful session ben bővítése - t**
        - ranzakció életciklus callbackekhez
    - **Passzivitás kikapcsolása**
        - passzivitás helyett egyből destroy
    - **Távoli és lokális interfészek egyszerűbb kötése**

- **Timer Service bővítés**
  - lekérhetjük az aktív timer példányokat
  - Tulajdonoson kívül is elkérhetőek a Timerek
- **Biztonság bővítzés**
  - “**” szere: minden bejeltnekzett felhasználó
  - Egyszerűbb szerepmegadások
- **Beágyazható EJB konténer bővítés**
  - AutoClosable
- **Marker interfész:**
  - nincsen benne metódus, annotációk vizsgálatával dönti el, milyen metódusokon keresztül kell üzeneteket továbbítani
- **API groups**
  - EJB lite-hoz hasonlú gyűjtemény
  - annál bővebb

- **JPA 2.1**
  - **Lekérdezések bővítése**
    - Downcasting: leszármazott osztály tipusára castolás
    - távoli eljárások meghívására támogatás
    - Outer join kibővítése
    - DB vagy JPA specifikus függvények hívása
    - Többes módosítása Criteria query-vel is
    - Futási időben definiálható @NamedQuers
    - Natív lekérdezések fejlettebb kezelése
    - Konverterek: oszlop érték és attribútum közé szúrható konvertálás, típuskonvertálás automnatikusa
  - **Entitás gráfok**:
    - lekérdezések fetch viselkedése hangolható
    - entitás gráfokat kapunk
    - Query hintek grád entitás használatára
  - **Nem szinkronizált perzisztenciakontextus**
    - Nyilvántarta a változásokat, de csak akkor flushol amikor tranzakcióhoz kötünk
  - **Sémagenerálás**
    - szabványos séma létrehozás
    - annotációk alapján, szkriptek meghívásával
    - séma kezelés, mi alapján hozza létre
    - DDL szkriptek
    - @Index: elsődleges kulcs
    - @ForeignKey, konstraint-et hoz létre automatikusa
- **CDI 1.1**
  - **Interceptorok globális engedélyezése**
    - prioritás, prioritási szintek
  - **Dekorátorok beépített beanekhez**
    - korábban csak saját interfészhez lehetett
  - **EvetMetadata**
    - más eseménykezelővel való integráció támogatásához
  - **@TransientReference**

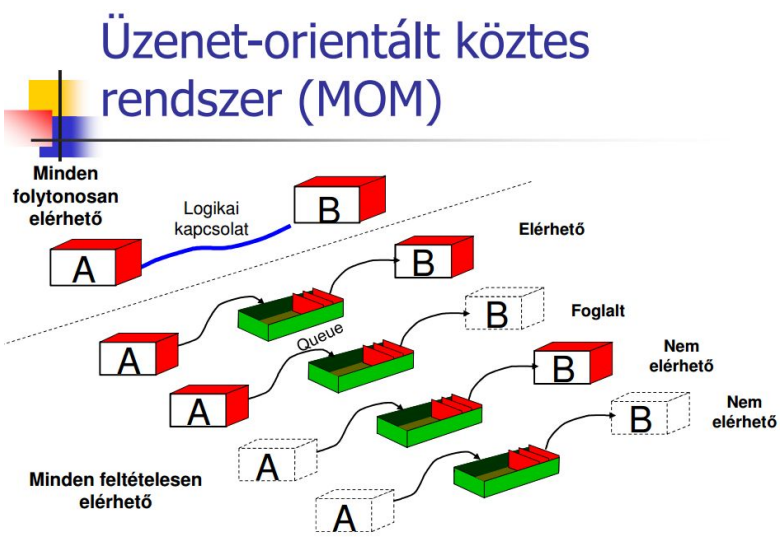
        - injektált változót ha nem akarjuk szerializálni, csak meghívni, ergo nem akarjuk letárolni akkor így kihagyható ebből a lépésből
    - **Scope inudlás - leállás kezelése**
        - annotációval
    - **Aktivitás**
        - nem kötelező a beans.xml -> de ilyenkor csak scope annotációkkal foglalkozik
        - kizárhatunk osztályokat
            - ezt is lehet annotációval, és beasn.xml-el
    - **Integráció más keretrendszerekkel**
        - Bean-ek CDI kontextuson kívüli létrehozása
        - CDI konténer programozott elérése
        - annotáció nélküli bean-ek behúzása
            - programozottan
- **Interceptors 1.2**
    - **@AroundConstruct**
    - Egyszerásített metódus szignatúrák
- **JTA 1.2**
    - **@Transactional:**
        - Deklaratív tranzakciókezelés nemcsak EJB-kre, hanem bármilyen CDI managed beanre
        - visszagörgetést kiváltó események paraméterként definiálhatók
    - **@TransactionScoped**
        - Aktuális JTA tranzakciókhoz köti a CDI bean élettartamát
- **Bean Validation 1.1**
    - **CDI integráció**
        - validátorok életciklusát is tudják kezelzni
    - **Metódus, és konstruktor validáció**
    - **Hibaüzenetek EL kifejezésekkel**
        - lokalizáció
- **JAX-RS 2.0**
    - **KLIENS OLDALI API**
        - eddig alacsony szinten volt, nem lehetett EntityProvidereket használni,
        - meg az intecepotorkat sem
        - mostmár de.
    - **Filterek és interceptorok**
        - Szervlethez hasonló célok
        - Kliens és szerver oldalon is már
        - **Filterek:**
            - nem kell explicite hívni a szűrőket, automtizált
        - **Interceptorok**
            - EntityProvidereket szakítják meg
            - Kliens és szerver oldalakon is
    - **Aszinkron feldolgozás**
        - **Szerver oldalon:**
            - Webkonténer IO szálját tehermenteysíti
        - **Kliens oldalon:**
            - Szolgáltatás meghívása szinkron módon

    - **Hypermedia**
        - A hypermedia a REST elvek továbbfejlesztése, mely szerint a REST kérés válaszában legyenek linkek, amelyek a visszaadott entitással végezhető műveleteket írják le
            - önleíró API
- **Servlet 3.1**
    - **Nem blokkoló IO**
        - eddig az volt, ha a Respons írást kértük
        - ReadListener, WriteListener
        - ServletInputStream, ServletOutputStream
        - csak aszinkron feldolgozás
    - **Upgade**
        - HTTP1.1 upgrade: Websocket is ezt használja
            - másik protokollra lehet váltani kommunikáció során
    - **Biztonság**
        - Session ID megszerzése -> para, megszemélyesítéses támadás
        - ezért új interfész: sessionIdListener -> sessionIdChanged?
        - “**” szerep, akár csak EJB-nél, minden bejelentkezett felhasználó
- **JSF**
    - **HTML5 támogatás**
        - eddog el akarta rejteni a részleteit
        - mostmár inkább minél jobban beépíteni
        - standard HTML tag-ek, csak szükség esetén egészítjük ki őket
        - HTML5-ös böngészők számára értelmezhet attribútumok, JSF nem törődik velük
    - **JSF tagek helyett standard HTML tagek**
        - speciális input tagek, HTML5 ből
    - **Resource Libary Contract**
        - Facelets template mechanizmusának továbbfejlesztése
            - template fájlok
        - contract = template + beszúrási pontok + erőforrások
    - **Face Flows**
        - összetartozó oldalak összekapcsolása, varázslók, workflow-k
        - olyamsmi, mint egy metódus,
            - alkalmazás bármelyik pontjáról hívhatóak
            - egymást is hívhatják a flow-k
            - egy belépési pont, bemenő paraméter, visszatérési érték
            - saját scope-ban tárolt bean-ek,
        - navigáció így már flow csomópontokra is értelmezettek
            - különféle csomópotn típusok
        - flow generálás XML-ben
    - **Stateless views:**
        - ha nem akarjuk, hogy a view-höz állapot rendelődjön
            - jobb teljesítmény
            - vagy klaszterezett környezet
- **EL 3.0**
    - **Kiterjesztés**
        - Használja a JSP, JSF, CDI és már a Bean Validation is

- **Új operátorok**
  - +=, string összefűzés
  - =: értékadás
  - egy kifejezésbe többb utasítás ;:
- **Saját konverterek**
- **Statikus metódusok és mezők**
- **Lambda kifejezések**
  - JAvaSE8 szintaxisa
  - Névtelen függvény
  - (x)->x+1
  - azinnal kiértékelődik, aztán eldobjuk a kifejezést
  - elnevezhető, rekurzív, stb.
- **Collection létreghozása**
  - set: {1,2,3} stb.
  - Collection műveletek, Stream, lamdba kifejezések stb.

18 Java Message Service

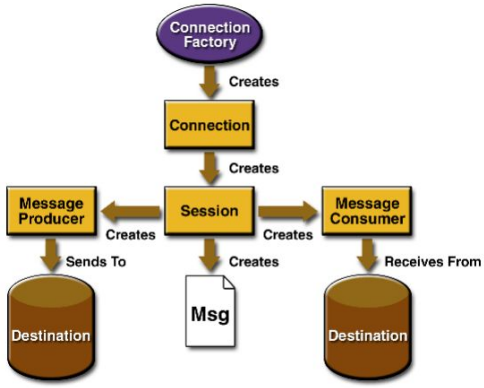
- Integrációs kényszer -> sok rendszer akar együtt működni
- **Integrációs környezet**
  - eltérő interfészek, eltérő platformok, eltérő kommunikációs protokollok, eltérő rendelkezésre állás, előre nem látható igények
  - valahogy a rendszer között kell a kapcsolat
  - **típusai**
    - **valós idejű**
      - rendszer a hozzá érkezett kérseket rövid időn belül kezeli
    - **szinkron üzenetváltás**
      - request-reply, blokkoló hívások
    - **aszonkron üzenetváltás**
      - requeast-reply, nem blokkoló hívások
- **Üzenet-orientált köztes rendszer MOM**
  - aszinkron, lazán csatolt, rendelkezésre állástól függelten,
    - de tranzakcióbiztos
  - közvetített üzenetek **biztos** egyszeri továbbítása, de csak egyszeri
  - alkalmazások eseményvezérelt üzemeltetésére
  - áttekinthető nredszer
  - platformfüggetlen, egyszerű API



- **Message Que-k**
- **Que manager**
- **Message-ek**

- **JMS Api**
  - kezdetben MOM rendszerek elérhetővé tétele
  - JEE1.3 óta van,
    - EJB-, webkomponensek és alkalmazás kliensek küldhetnek, szinkron módon foghadhatnak
    - Message Driven bean-ek: aszinkorn üzenet fogadásra
    - üzenetek küldése és fogadása elosztott tranzakcim belül is történhez
  - **JMS architektúra**
    - **Provider**
    - **Client**
    - **Administered objects**
  - **Kommunikációs modell**
    - **Point-to-Point**
      - középen van a Que, két oldalán a kliensek
    - **Publish-subscribe**
      - különböző topicocra lehet feliratkozni, amibe ha valaki publikál, megkapjuk
  - **Programozási modell**
    - **üzenet**
      - részei: fejléc, properties, törzs
      - **fejléc mezők:**
        - azonosító, célobjektum, perzistens-e? lejárati idő, prioritás, küldési idő, hivatkozott üzenet, válasz célojbaktumra, üzenet típusa, újraküldő jelző flag
      - **properties**
        - fejléc bővítése saját mezőkke
        - általános típusokra
      - **törzs típusok**
        - Message, BytesMesage, MapMessage, ObjectMesage, StreaMesage, TextMessage

Programozási modell



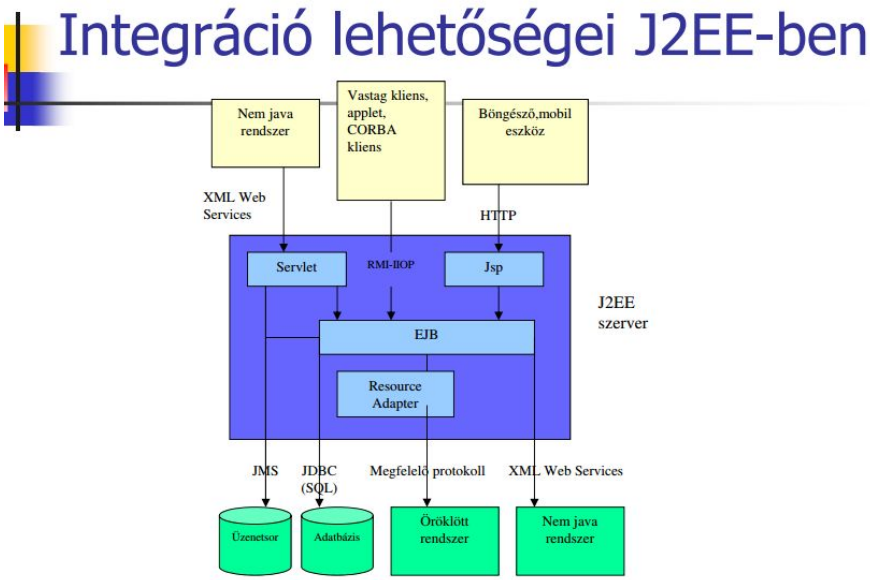
- **ConnectionFactory**
  - konfigurációs paraméterek
  - kapcsolatok létrehozása
  - elrejti a priovider specifikumait
  - JNDI Namig XContext-ben tárolódik
  - Elkérni lehet a QueueConnectionFactory-tól egy példányt
  - létre lehet hozni, Queue-t és Topiic connection-t
- **Connection**
  - egy virtuális kapcsolat a providerrel
  - Használata: a kapcsolatot elindítjuk start()
  - lezárjuk stop()
  - csak közte lehet üzeneteket fogadni
  - Használata pl. egy Session elkérése
- **Session**
  - egyszűlő, tranzakciós kontextus
  - előállít Producert, Consumert, Message-et.
- **Destination**
  - lehet Adminisztrált, vagy ideiglenes
- **QueueSession**
  - Point - to - point
  - lehet küldő, lehet fogadó
- **TopicSession**
  - publish-subscrube
  - lehet publisher és subscripbert létrehozni
  - TopicSubscriber
    - Non-durable: csak azokat az üzenetek kapjameg, amelyeket akkor publikálnak amikor aktív
    - Durable: inaktív állapota alatt publikálta
- **MessageListener intefrész**
  - saját aszinkron üzenetfeldoglozó osztály
  - egyszáló használat!
- **MessageSelector string**

        - üzenetek szűrését teszi lehetővé
        - JMS provider gondoskodi a szűrésről
- **Message-Driven Bean**
    - aszinkron üzenet-feldolgozást végez
    - nincs remote-locatl intefésze, csak egyetlen fájl
    - Szinkron módon, közvetlenül nem meghívható
    - stateless
    - fogadhat üzenetet:
        - Queue-ból
        - Topic-ból
    - beállítható annotációval, telepítésleíróval
    - beállítások: Trtranzakcionalitás, messageSelector, DestinationType(Queue, Topic)
    - még beállítások: ConnectionFactorsy, Poolméretek, Durable esetén a subscriber egyedi neve
    - **egy kezelendő probléma**
        - **Poison message**
            - amit a MDB nem tud feldolgozni, mert
                - üzenet korrupt,
                - formátuma hibás
                - feldolgozásakor hiba
            - Célt tévesztő üzenet
            - onMessage nem dob exception-t…
            - **Kezelése**
                - **container managed tranzakció esetén**
                    - Rollback
                - **bean managed tranzakció esetén**
                    - kiolvasás nem része a tranzakciónak, így áthelyezhetjük egy másik ques-ba, vagy eldobjuk az üzenetet
    - **Válaszadás lehetősége**
        - több kliens szólít meg egy alkalmazást
        - minden kérés ugyan abba a queue-ba megy
        - minden alkalmazás csak a neki szóló üzeneteket szeretné látni
        - **megoldás:**
            - válasz-topic, minden kliens message selectorral olvassa
            - vagy mindenkienk saját queue-ja legyen
    - **Best Practices**
        - különítsd el az üzleti logikát a MDB-től
        - Request-Reply objektumokat ne keverjük
        - MDB legyen tényletg állapotmentes
        - ne függjön az alkalamzás az üzenetek sorrendjétől
        - Üzenet-méret korlátokkal számolj
        - Alkalmazások közötti kommunikációára -> XML üzenetek
        - Kerüld a hosszú lefutású tranzakciókat MDB-től
        - készülj fel a poison message-ek kezelésére
    - **JMS a JavaEE-ben**
        - a fejlődéssel főleg annotációk, rövidítés, bolerplate kódok kiszedése
        - régi apit is lehet használni, de körülményes mindent létrehozni

        - JMS2.0 JMSContextet injektálása
        - üzenet fogadások egyszerűbbek
        - Erőforrásokat lehet lértrehozni, JNDI
            - ezek lehetnek Queueu-k
            - alkalmazásban is létrehozhatóak, majd beregisztrálja az alkalmazásszerver
        - különféle hasznos dolgok: sikertelen kézbessítések számának lekérdezése, delivery Delay beállítás, stb.

19 Java Connector Architecture

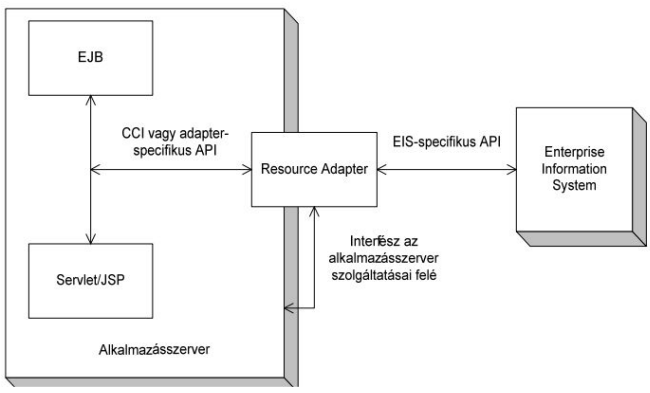
- **Integráció jelentőssége**
    - Egy vállalatnak több alkalmazása van
        - mindegyik máskor készül, más platform, minden más balbal
    - ezek működjenek együtt
        - olcsóbb, mint mindent egy platformra újraírni



    - **1. Egyszerű HTTP:**
        - állapotmentes
        - nem OO
        - nem tranzakcionális,
        - elég gagyi.
    - **2. DB szintű integráció**
        - nem túl szoros együttműködésre
        - amúgy végső soron üzenetküldözgetésbe torkollik
    - **3. RMI Remote Method Invocation**
        - OO, csak Java
        - JavaEE környezetben minkább RMI-IIOOP
    - **IIOP Internet Inter-ORB Protocol**
        - OO, nyelvfüggelten
        - csak CORBA laklamazások számára

        - EJB-k alapból hívhatóak IIOP-on
    - **JMS Java Message Service**
        - aszinkron, üzenetorientált
        - nyelv, platformfüggelten
        - message-driven beanekkel egyszerű, hatékony az üzenetek fogadása
        - ki tudja használni a J2EE szerver middleware szolgáltatásait
    - **Webszolgáltatások**
        - szinkron, vagy aszinkron
        - nyelv- és platformfüggetlen
        - kevésbé robosztus
        - ezek még nincsenek teljesen szabványosítva
    - **J2EE Connector Architecture JCA**
        - Egy JEE alkalmazásszerver tetszőleges információs rendszerre való együttműködésnek megvalósítására
        - saját fejlesztésű rendszerekre pl.
        - Megoldandó feladatok
            - csak natív hvásokkal JNI vagy socketen kereszütl elérhető rendszerek -> biztonságossá , robosztussá kell tenni
            - alkalmazásszererrel egysüttműködés
            - alékalmazásszerverek közötti hordozhatóság kérdése
        - **resource Adapter RA**
            - JCA központi eleme, 3 dologgal működik együtt a JCA
                - 1. klienssel (Servlet, EJB)
                - 2. alkalmazásszerver szolgátatásaival
                - 3. az EIS-sel
            - fizikailag: .calss fájlok és egy telepítésleíró
            - RA futhat más alkalmazásszerveren kívül is, ekkor a kliens standard Java app

Resource Adapter (RA)



            - Interfészei:
                - EIS felé EIS specifikus
                - Kliens felé
                    - adapter-specifikus
                    - Common Client Interface CCI, általános api
                - alkalmazásszerver szolgáltatásai felé:

        - System Contract-ok!
        - LifeCycle, Conection, Transaction, Security, Work message stb. manadgement
    - Egy contract megvalósíátsa két dolgot jelent
        - java interfészeket megvalósító osztályok
        - megfelelő telepítésleírók, annotációk
    - **RA interfészek**
        - **Life Cycle Management**
            - ha ilyen van, akkor az alkalmazásszerver értesíteni fogja őt
            - különböző szolgáltatásokat egy contexten kereszül lehet elérni
                - timer szolgáltatás
                - work manager
        - **Connection Management**
            - kapcsolatok újrahasznosítása
            - minden alkalmazásszervern futó alkalmazás megoszthatja a kapcsolatokat
            - kliens oldali kapcsolat megkülönböztetve az EIS oldalitól
            - contract-okon kereszült
        - **Security Management**
            - autentikáció EIS felé, két módon
        - **Transaction mangement**
            - lokális és globális tranzakció
            - konténer által kezelt tranzakció vs. programozott traznakció
        - **Work management**
            - új szál indítása alkalmazásszerver által menedzselt környezetben nem triviális -> EJB előírások ellenőrzése
            - alkalmazásszerver futassa az új szálatkat
            - Work példányoakt d át
        - **Message In-flow**
            - üzenetek
- **Anntoációk**
    - **JEE6 féle, teljesen felüírhatóak**
        - @Connecvtor, @configProperty @Acitavion stb.
- **Összefoglalás**
    - **Mikor válasszuk a JCA -t integrációs megoldásként?**
        - ha nem akarjuk módosítani az EIS-t amelyikkel együtt kell működni
        - robosztusl, alkalmazásszerver szolgáltatásait kihasználó megoldásra van szükség
        - elterjed EIS-hez akarunk csatlakozni,-> lesz hozzá kész connector :P

20. Biztonsági megfontolások

- **Alapfogalmak**

    - **Autentikáció:** a felhasználó azonosítása, felh.név + jelszó
    - **Autorizáció:** milyen műveletekhez férhet hozzá a felhasználó?
    - **Adat integritás:** adatok módosítása detektálható
    - **Adatok biztonsága:** csak az arra jogosultak tudnak hozzájuk férni
    - **Letagadhatatlanság:** bizonyítható ha valaki valamilyen műveletet elvégzett
- **Biztonsági kockázat**
    - Sikeres támadás valsége * támadás által okozott veszteség
    - 0 kockázat akkor lehet, ha
        - 0 értéket véd a rendszerünk
        - 0 a sikeres támadás valsége
    - tehát nincs tökéletes biztonság.
    - Célunk: a sikeres támadás költsége haladja meg a vele szerezhető hasznot
- **Java 2 Security**
    - Security a kezdetekben
    - főleg appletek-> user megóvása a böngészőben futó alkalmazásoktól, homokozóban
    - helyi gépen futó app viszont bármit megtehetett, amit a JVM
    - appletek terjedése -> kellett kilépni a homokozón túlra
    - jdk 1.1 signed apple
    - java 2 security: finomhangolás
        - még mindig a felhasználó megvédése
        - Jogosultságellenrzés: SecurityManadgement checkpermission
        - futás időben
        - exceotiont dobtak, szálak jogosultságát vizsgálta
    - **AccessControlContext:**
        - egy szál ACC-jébe folyamatosan kerülnek ki be a domain-ek, stack apaján
        - ezen ellenőrzünk
    - **Policy:**
        - hol definiálhatók a tényleges jogok?
        - A singleton Policy objektumban
        - **Policy fájl**
            - felüldefiniálható, alapértelmezett
            - JEE vonatkozás: alkalmazásszerver nem fér hozzzá mondjuk fájlokhoz, amíg nem kapja meg a jogot hozzá,
    - **Összefoglalva**
        - java.security csomag
        - fontos osztályok:
            - SecurityManager
            - AccessControlContext
            - ProtectionDomain
            - Perimission
            - Policy
            - SecureClassLoader
- **JAAS**
    - egyre inkább általános célú, kliens-szerver alkalmazások fejlesztésére használták a java-t
    - **megfordult az irány: az alkalmazás funkcióit kell védeni a felhasználóktól**
    - **Java Authentication and Authorization Service**

- J2EE 1.3 óta
- ez is a security alapcsomagban

- **Támadási módszerek**
  - **Hiába állítunk be autentikációt és megfeleő autorizációt a komponenseinkben**
    - **1.** mert marad az **SQL injection**
      - SQL injection: a stringgel játszás
      - védekezés: PreparedStatement használata
      - a setString-et úgy implementálja a JDBC driver, hogy escapeli a paramétert
      - O-R technológiák mindegyike ezt használja már EJB 2.x, entitások, stb.
    - **2.** mert marad a Cross-site scripting **XSS**
      - Dinamikus webappoknál
      - HTML formban az input mezőbe beírt szöveg megjenei egy újonnan generált HTML oldalon
      - A támadó: javascript szöveget ír be inputként
        - ez a DB-ben tárolódik
        - más user megnézi, az ó HTML-jébe az adabázisból kiolvasva
        - már javascript generálódik
        - ami az ő gépükön meg lefut. PL Session ID-t elküldi a támadónak
      - ha nem a DB-ben tárolódik az input, hanem egyből megjelni a válaszoldalon, akkor meg ráveszi a usert hogy elmenjen egy random olrara, másoljon be egy javascriptet (emailt küldd, amiben egy link va, az URL get paramétere)
      - védekezlés:
        - kimenet HTML escapelése <> helyett &lt
        - JSP oldalon a c:cout, JSF oldalon a f:outputText pont ezt teszi
        - ehezebb megoldani, ha saját tag-eket akarunk használni
  - **Infrastruktúra biztonsága**
    - teljesen biztonságosra megírt alkalmazás sem ér semmit, ha **nem biztonságos a környezet amiben fut**
    - **Tipikus hibák**
      - default jelszavak bennhagyása
      - alkalmazásszerverre alapértelmezetten települő egyszerű minialkalmazás XSS támadási felülettel
      - leggyakrabban a JNDI providerbe alapértelmezetten bárki regisztrálhat bármit -> a támadó saját EJB-ket ad be…
      - nem elég erős biztonsági algoritmus pl SSL esetén
      - OS, alkalmazásszerver, adatbázisszerver hiányosságai
      - nem átgondolt jogosutlságok a fájrendszer, alkalamzásszerver, OS, adatbázis zerver szinten
      - nem megfeleő tűzfal
      - nyitva fejeltett portok
  - Eghyéb biztonsággal kapcsolatos JAVA API-k
    - **Java Cryptohrapgy Extension JCE**
      - titkosítási algoritmusok
      - kulcsgenerálás, kulcscsere
    - **Java Secure Sockets Extension JSSE**
      - SSL támogatás

- **Java generic Security Services Java GSS-API**
  - alkalmazás szintű protokoll, token alapú,
- **Simple Authentication and Security Layes SALS**
  - LDAP, IMAP használja