J2EE for begyndere

Denne artikel introducerer J2EE.

**Den fortæller lidt om de forskellige teknologier
i J2EE og kommer med små simple eksempler.**

Den forudsætter kendskab til Java og generel system udvikling.

Skrevet den **11. Feb 2009** af **arne_v** I kategorien **Programmering / J2EE** | ★★★☆☆

Historie:
V1.0 - 21/02/2004 - original
V1.1 - tilføj et par referencer og en note om det nye navn JEE

Hvad er J2EE

J2EE består af 4 hoved komponenter:

- servlets
- JSP (Java Server Pages)
- EJB (Enterprise Java Beans)
- JCA (Java Connector Architecture)

Servlets er rene Java kode moduler i stil med ISAPI og NSAPI.

JSP er HTML sider med embbeded kode ligesom ASP og PHP (det er bare Java kode i <% %>)

EJB er forretnings komponenter.

JCA er en standard for forbindelser til eksterne systemer.

Der eksisterer følgende sammenhæng mellem versionerne af de forskellige teknologier:

J2EE 1.2 = servlet 2.2 + JSP 1.1 + EJB 1.1
J2EE 1.3 = servlet 2.3 + JSP 1.2 + EJB 2.0 + JCA 1.0
J2EE 1.4 = servlet 2.4 + JSP 2.0 + EJB 2.1 + JCA 1.5

Jeg vil gå let hen over servlet og JSP, da jeg har beskrevet dem i en anden artikel.

Se http://www.eksperten.dk/artikler/28.

Og for dem som vil videre se også http://www.eksperten.dk/artikler/743.

Fra og med næste version skifter J2EE navn til JEE og versionsnumre fra n.m til n. Det betyder at J2EE 1.4 vil blive afløse af JEE 5.
Jeg vil bruge J2EE i resten af artiklen.

Servere

Man skal selvfølgelig have en server. Der er mange muligheder.

Kommercielle:
- IBM WebSphere (WAS)
- BEA WebLogic (WL)
- Oracle OC4J
- Borland Enterprise Server (BES)
- Sun ONE Applicaion Server
- Macromedia JRun.
- Pramati

Gratis:
- JBoss
- Jonas
- Reference Implementation

De 3 mest udbredte idag er WebSphere, WebLogic og JBoss. Da JBoss er den eneste gratis af de 3 vil den være et oplagt valg, hvis man vil igang med J2EE.

Reference implementation i J2EE 1.3 var forfærdelig. SUN har brugt en neddroslet version af deres Sun ONE Applicaion Server som Reference Implementation i 1.4, spm formentlig er bedre selvom Sun ikke har noget godt ry med hensyn til app servere.

Arkitektur
J2EE løsninger er typisk 4 tier web applikation:
browser ---- JSP/Servlet container ---- EJB container ---- database server

Men andre løsninger ses en gang imellem.

3 tier applikation:
client app ---- EJB container ---- database server

4 tier web service:
client app ---- JSP/Servlet container ---- EJB container ---- database server

5 tier web applikation:
browser ---- JSP/Servlet container ---- EJB container ---- EJB container ---- database server

Forskellige EJB's
Der er 3 forskellige slags EJB:
- session bean = synchroneous service (client kalder og der returneres når færdig)
- message driven bean = asynchroneous service (messages fra kø processes)
- entity bean = persisteret record

Der er 2 forskellige slags session beans:
- stateful session bean = session bean hvor alle kald fra en client går til samme bean instans
- stateless session bean = session bean hvor kald fra client går til vilkårlig bean instans

Session beans kan også opdeles efter om de bruger:
- CMT = Container Managed Transactions = serveren styrer transaktioner
- BMT = Bean Managed Transactions = bruger koden styrer transaktioner

Der er 2 forskellige slags entity beans:
- CMP = Container Managed Persistence = serveren henter fra og gemmer til databasen
- BMP = Bean Managed Persistence = bruger koden henter fra og gemmer til databasen

EJB's
Der er mange interfaces/klasser involveret i en EJB.

Dem som du koder:
  remote interface = de metoder som både kan tilgåes lokalt og fra andre systemer
  local interface = de metoder som kun kan tilgåes lokalt
  remote home interface = de metoder som kan bruges til at oprette/finde/slette
                          både lokalt og fra andre systemer
  local home interface = de metoder som kan bruges til at oprette/finde/slette
                         kun lokalt
  bean klasse = selve implementeringen

Dem som containeren genererer:
  remote object klasse
  local object klasse
  remote home object klasse
  local home object klasse
  stub/skeleton klasser for remote object klasse
  stub/skeleton klasser for remote home object klasse

Når man bruger en EJB så foregår det som:
1) lookup i JNDI returnerer et remote/local home interface (med en klasse bagved)
2) kald af metode i dette opretter/finder et remote/local interface (med en klasse bagved)
3) kald af metoder i dette dispatches videre til din bean klasse

De compilede .class filer pakkes ned i en jar fil typisk kaldet xxx-ejb.jar, som
så deployes til serveren.

Udover .class filer indeholder den også:
  META-INF/ejb.jar.xml = standard deployment descriptor
  META-IF/*.xml = server specifikke deployment descriptorer

JCA adapterer
JCA er en generalisering af JDBC adgang til databaser.

D.v.s. at en JCA adapter bruges til at interface et eksternt system
som kan være en database eller et regn skabs system eller noget helt
tredie.

Standardiseringen af den slags interfaces er en af de ting som er
unikke i J2EE.

Derudover bliver JCA adaptere også lidt brugt som skralde spand fordi
der er nogen restriktioner på EJB's:
  - de må ikke starte tråde
  - de må ikke lytte på sockets
  - de må ikke bruge JNI
  etc.
og JCA adaptorer må godt alt dette.

Simple kode eksempler
Vi laver er simpelt eksempel med 1 session bean kun med remote
tilgang og 1 entity bean kun med local adgang. Entity bean
laves med CMP. Eksemplet er ikke et realistisk J2EE eksempel
men illustrerer J2EE kode.

TestSessionBean.java = bean klasse

```
package test;

import java.util.*;

import javax.naming.*;
import javax.ejb.*;

/**
* TestSessionBean.
*/
public class TestSessionBean implements SessionBean {
    private SessionContext sessionContext;
    // diverse standard som skal være der men som man sjældent bruger
    public void ejbCreate() {
        return;
    }
    public void ejbRemove() {
        return;
    }
    public void ejbActivate() {
        return;
    }
    public void ejbPassivate() {
        return;
    }
    public void setSessionContext(SessionContext sessionContext) {
        this.sessionContext = sessionContext;
    }
    // gem
    public void save(int f1, String f2) {
        try {
            TestEntityLocalHome tehome = lookup();
            tehome.create(new Integer(f1), f2);
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
    // hent F2 for specifik F1
    public String getOne(int f1) {
        try {
            TestEntityLocalHome tehome = lookup();
            TestEntityLocal tebean = tehome.findByPrimaryKey(new Integer(f1));
            return tebean.getF2();
        } catch(Exception ex) {
            ex.printStackTrace();
            return null;
        }
    }
    // hent alle F1
    public List getAll() {
        List lst = new ArrayList();
        try {
```

```
            TestEntityLocalHome tehome = lookup();
            Iterator it = tehome.findAll().iterator();
            while (it.hasNext()) {
                TestEntityLocal tebean = (TestEntityLocal)it.next();
                lst.add(tebean.getF1());
            }
        } catch(Exception ex) {
            ex.printStackTrace();
        }
        return lst;
    }
    // lookup TestEntity
    private TestEntityLocalHome lookup() throws Exception {
        // create JNDI context
        Context ctx = new InitialContext();
        // lookup TestEntity local home interface
        Object temp = ctx.lookup("ejb/LocalTestEntity");
        // cast fra object til local home interface
        TestEntityLocalHome tehome = (TestEntityLocalHome)temp;
        return tehome;
    }
}
```

TestSession.java = remote interface

```
package test;

import java.util.*;
import java.rmi.*;

import javax.ejb.*;

/**
* Remote interface for TestSession EJB.
*/
public interface TestSession extends EJBObject {
    // gem
    public void save(int f1, String f2) throws RemoteException;
    // hent F2 for specifik F1
    public String getOne(int f1) throws RemoteException;
    // hent alle F1
    public List getAll() throws RemoteException;
}
```

TestSessionHome.java = remote home interface

```
package test;

import java.rmi.*;
```

```
import javax.ejb.*;

/**
* Home interface for TestSession.
*/
public interface TestSessionHome extends EJBHome {
    // opret remote object
    public TestSession create() throws CreateException,RemoteException;
}
```

TestEntityBean.java = bean klasse

```
package test;

import javax.ejb.*;

/**
* TestEntityBean.
*/
abstract public class TestEntityBean implements EntityBean {
    private EntityContext entityContext;
    // create
    public Integer ejbCreate(Integer f1, String f2) throws CreateException {
        setF1(f1);
        setF2(f2);
        return null;
    }
    public void ejbPostCreate(Integer f1, String f2) throws CreateException {
    }
    // diverse standard som skal være der men som man sjældent bruger
    public void ejbRemove() throws RemoveException {
    }
    public void ejbLoad() {
    }
    public void ejbStore() {
    }
    public void ejbActivate() {
    }
    public void ejbPassivate() {
    }
    public void setEntityContext(EntityContext entityContext) {
        this.entityContext = entityContext;
    }
    public void unsetEntityContext() {
        this.entityContext = null;
    }
    // get og set F1
    public abstract Integer getF1();
    public abstract void setF1(Integer f1);
    // get og set F2
    public abstract String getF2();
    public abstract void setF2(String f2);
```

```
}
```

TestEntityLocal.java = local interface

```
package test;

import javax.ejb.*;

/**
* Local interface for TestEntity.
*/
public interface TestEntityLocal extends EJBLocalObject {
    // get and set F1
    public Integer getF1();
    public void setF1(Integer f1) ;
    // get and set F2
    public String getF2();
    public void setF2(String f2);
}
```

TestEntityLocalHome.java = local home interface

```
package test;

import java.util.*;

import javax.ejb.*;

/**
* Local home interface for TestEntity.
*/
public interface TestEntityLocalHome extends EJBLocalHome {
    // opret record
    public TestEntityLocal create(Integer f1, String f2) throws
CreateException;
    // find record udfra primary key
    public TestEntityLocal findByPrimaryKey(Integer f1) throws
FinderException;
    // find alle records
    public Collection findAll() throws FinderException;
}
```

ejb-jar.xml = deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
```

```
<ejb-jar >
   <enterprise-beans>
      <session >
         <!-- definer interfaces, klasser m.v. for TestSession -->
         <ejb-name>TestSession</ejb-name>
         <home>test.TestSessionHome</home>
         <remote>test.TestSession</remote>
         <ejb-class>test.TestSessionBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
      </session>
      <entity>
         <!-- definer interfaces, klasser, primary key, persisterede felter
m.v. for TestEntity -->
         <ejb-name>TestEntity</ejb-name>
         <local-home>test.TestEntityLocalHome</local-home>
         <local>test.TestEntityLocal</local>
         <ejb-class>test.TestEntityBean</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.Integer</prim-key-class>
         <reentrant>False</reentrant>
         <cmp-version>2.x</cmp-version>
         <abstract-schema-name>TestEntity</abstract-schema-name>
         <cmp-field >
            <field-name>f1</field-name>
         </cmp-field>
         <cmp-field >
            <field-name>f2</field-name>
         </cmp-field>
         <primkey-field>f1</primkey-field>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Deployment
Vi vil nu deploye på JBoss så vi tilføjer 2 JBoss specifikke deployment descriptorer.

jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">
<jboss>
   <enterprise-beans>
      <session>
         <!-- definer remote JNDI navn for TestSession -->
         <ejb-name>TestSession</ejb-name>
         <jndi-name>ejb/TestSession</jndi-name>
      </session>
      <entity>
         <!-- definer local JNDI navn for TestEntity -->
         <ejb-name>TestEntity</ejb-name>
```

```
         <local-jndi-name>ejb/LocalTestEntity</local-jndi-name>
      </entity>
   </enterprise-beans>
</jboss>
```

jbosscmp-jdbc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbosscmp-jdbc PUBLIC "-//JBoss//DTD JBOSSCMP-JDBC 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_2.dtd">
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <!-- definer tabel navn og felt navne i databasen for TestSession -->
         <ejb-name>TestEntity</ejb-name>
         <datasource>java:/TestMySQL</datasource>
         <datasource-mapping>mySQL</datasource-mapping>
         <table-name>T1</table-name>
         <cmp-field>
            <field-name>f1</field-name>
            <column-name>F1</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>f2</field-name>
            <column-name>F2</column-name>
        </cmp-field>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Efter build indeholder test-ejb.jar nu:

```
     0 Fri Feb 20 23:11:00 CET 2004 META-INF/
   107 Fri Feb 20 23:10:58 CET 2004 META-INF/MANIFEST.MF
     0 Fri Feb 20 22:26:28 CET 2004 test/
  1038 Fri Feb 20 22:19:52 CET 2004 test/TestEntityBean.class
   264 Fri Feb 20 22:19:52 CET 2004 test/TestEntityLocal.class
   410 Fri Feb 20 22:19:52 CET 2004 test/TestEntityLocalHome.class
   287 Fri Feb 20 22:19:52 CET 2004 test/TestSession.class
  1774 Fri Feb 20 23:11:00 CET 2004 test/TestSessionBean.class
   223 Fri Feb 20 22:26:28 CET 2004 test/TestSessionHome.class
  1419 Fri Feb 20 22:49:46 CET 2004 META-INF/ejb-jar.xml
   575 Fri Feb 20 22:33:34 CET 2004 META-INF/jboss.xml
   771 Fri Feb 20 22:34:02 CET 2004 META-INF/jbosscmp-jdbc.xml
```

Vi laver nu en data source deployment descriptor.

testmysql-ds.xml

```
<datasources>
  <local-tx-datasource>
    <jndi-name>TestMySQL</jndi-name>
    <connection-url>jdbc:mysql://localhost/Test</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name></user-name>
    <password></password>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>100</max-pool-size>
  </local-tx-datasource>
</datasources>
```

Og så kopierer vi testmysql-ds.xml og test-ejb.jar til JBoss deploy directory
og så er vi klar til at teste.

Test.java = test client

```
package e_j2ee;

import java.util.*;

import javax.naming.*;
import javax.rmi.*;

import test.*;

/**
* Test client.
*/
public class Test {
    public static void main(String[] args) {
        // set properties for at kunne lave lookup til JBoss JNDI
        System.setProperty("java.naming.factory.initial",
"org.jnp.interfaces.NamingContextFactory");
        System.setProperty("java.naming.factory.url.pkgs",
"org.jnp.interfaces");
        System.setProperty("java.naming.provider.url", "localhost:1099");
        try {
            // create JNDI context
            Context ctx = new InitialContext();
            // lookup TestSession home interface
            Object temp = ctx.lookup("ejb/TestSession");
            // narrow fra object til local home interface
            TestSessionHome tshome = (TestSessionHome)
PortableRemoteObject.narrow(temp, TestSessionHome.class);
            // create
            TestSession tsbean = tshome.create();
            // create record
            tsbean.save(123, "ABC");
            // lookup alle records
            List lst = tsbean.getAll();
```

```
            for(int i = 0; i < lst.size(); i++) {
                int f1 = ((Integer)lst.get(i)).intValue();
                String f2 = tsbean.getOne(f1);
                System.out.println(f1 + " " + f2);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Dette var kun et lille bitte eksempel - der er mange andre muligheder i J2EE, som jeg ikke kan nå at komme ind på.

Hvad er J2EE godt til ?
J2EE har er et ry som værende "fint".

Og det er faktisk også en utrolig kraftfuld teknologi. Men det er også en kompleks teknologi. Og det er ikke nødvendigvis den rette løsning til alle opgaver.

Forhold som gør det relevant at anvende J2EE:

1) krav om horisontal skalerbarhed d.v.s. at der skal kunne tilføjes flere servere som bruger loadsharing

2) kompleks forretnings logik d.v.s. ikke kun simpel præsentation

3) så stort et system at kraftig seperation mellem præsentation, forretnings logik og persistering er absolut nødvendig

4) data i flere forskellige XA compliant databaser som skal opdateres atomisk

5) integration med andre eksterne systemer end databaser

Hvis flere mange af disse forhold er opfyldt så er J2EE et godt valg af teknologi, som minimere både udviklings og vedligeholdelses omkostninger.

Hvis næsten ingen af disse forhold er opfykdt, så er J2EE et teknologisk overkill, som vil give unødvendige omkostninger.

Glem alle argumenter om at "J2EE er langsomt", "entity beans er langsomme". Det har intet med virkeligheden at gøre. Udviklere som ikke forstår komplekse distribuerede systemer kan få enhver teknologi til at køre langsomt. Det er ikke teknologiens skyld. J2EE er en meget kompleks teknlogi. Det tager lang tid at sætte sig ordentligt ind i den. Hvis man starter et komplekst J2EE projekt uden den fornødne viden, så er der stor risiko for at projektet kommer i problemer. Men det ville det også med en anden teknologi.

Kommentar af ihtezaz d. 22. May 2004 | 1

Det helt godt Arne, den giver god indblik i den komplicerede teknologi. For J2EE er blandt de meste populere teknologier og godt at kunne håndtere.

Kommentar af medions d. 06. Mar 2004 | 2

Virklig god artikel! Du kan sq bare det der Arne!!

Kommentar af conrad d. 24. Feb 2004 | 3

Virkelig god artikel om et kompliceret område - særligt begrundelsen for at vælge J2EE synes jeg er rigtig god at have med

Kommentar af rudidanmark d. 25. Apr 2005 | 4

God og overskuelig artikel.
Savner en reference til den artikel omkring jsp og servlets som du beskriver i starten.

Kommentar af globulous d. 06. Feb 2006 | 5

Kommentar af funk-food d. 26. Apr 2004 | 6

Virkelig en god artikel fra dig endnu en gang synes det er meget godt forklaret

Kommentar af jenniferw d. 06. Sep 2005 | 7

God og informativ!