Lottery #gradle

JMS, Message Driven Bean, JMX, Singleton Session Bean

Óbudai Egyetem, Java Enterprise Edition

Műszaki Informatika szak

Labor 6

Bedők Dávid

2018-01-17

v1.0

Lottery

Ötös lottóhúzás eredményeinek kezelése

Feladat: hozzunk létre egy Enterprise Java alkalmazást, mely az ötös lottóhúzás (számok 1-től 90-ig, azonos valószínűséggel) heti eredményeit tárolja (időbélyeggel, aktuális nyereményalappal és a számot kihúzó személy nevével).

- A kihúzott számokat egy **JMS queue**-n keresztül fogja az alkalmazás megkapni (az öt számot egyben, vesszővel elválasztva szöveges formátumban) (machine-to-machine interface).
- RESTful interface-en keresztül legyen lehetőség az aktuális és az összes korábbi lottóhúzás adatainak lekérdezésére.
- A szolgáltatás tartsa karban, hogy az aktuális szabályok szerint milyen a nyereményalap eloszlása (pl. a három találatos szelvények a nyereményalap mekkora részére jogosultak).
- RESTful interface-en keresztül lehessen öt kihúzott számról megkérdezni, hogy az aktuális sorsolás figyelembe vételével a nyereményalap mekkora részéből remélhet nyereményt (és hogy egyáltalán nyertes szelvényről van-e szó).
- A szolgáltatást **szabványos management** (**JMX**) felületen keresztül lehessen finomhangolni (nyereményalap, számhúzó személy, nyeremények eloszlása).

Ismeretszerzés

lottery project

- JBoss Management Console
  - admin/guest felhasználó létrehozása
- **J**ava **M**essage **S**ervice (**JMS**)
  - JMS Desination
    - JMS Queue
    - JMS Topic (feladat nem érinti)
  - **M**essage **D**riven **B**ean (**MDB**) (listener)
  - JMS Client
- **J**ava **M**anagement e**X**tension (**JMX**)
  - Standard **M**anagement **Bean** (MBean)
  - jconsole
- Singleton Session Bean
  - speciális állapottal rendelkező Session Bean

Project stuktúra

Alprojektek, modulok

- **lottery** (root project)
  - **lot-webservice** (EAR web module)
    - A RESTful webservice-ek project-je (presentation-tier).
  - **lot-ejbservice** (EAR ejb module)
    - Üzleti metódusok (service-tier)
  - **lot-persistence** (EAR ejb module)
    - ORM réteg, JPA (data-tier)
  - **lot-jmsclient** (standalone)
    - JMS kliens alkalmazás
  - **lot-jmxclient** (standalone)
    - JMX kliens alkalmazás

Maven esetén egy **lot-ear** project is megjelenik.

A **JMS client** alkalmazás *classpath*-ához jelenleg nem lesz szükség pl. egy `lot-ejbserviceclient` projectre, mivel a kommunikáció szabványos (nem kell pl. remote interface), és szöveges üzeneteket fogunk küldeni (nem kellenek stub-ok). Utóbbi megléte és Object JMS message esetén megjelenne természetesen a `lot-ejbserviceclient` is, mely része lenne mind az EAR mind a JMS client *classpath*ának.

JBoss management console

admin felhasználó létrehozása

- http://localhost:9990/
- Használatához egy **admin** jogosultságú management user szükséges (BASIC AUTH)
  - [JBOSS-HOME]/bin/add-user.[bat|sh]
    - Management User (enter)
    - Username: admin (are you sure? yes)
    - Password: AlmafA11#
    - Enter, Yes, Yes, Enter

```
1 > [JBOSS_HOME]/bin/add-user.[bat|sh] admin AlmafA11#
```

  - [JBOSS-HOME]/standalone/configuration/mgmt-users.properties
- **guest** felhasználó létrehozása (később szükség lesz egy sikeresen authentikálható, de admin jogokkal nem rendelkező felhasználóra):

```
1 > [JBOSS_HOME]/bin/add-user.[bat|sh] -a -u jmstestuser
    -p User#70365 -g guest
```

  - [JBOSS-HOME]/standalone/configuration/application-users.properties
  - [JBOSS-HOME]/standalone/configuration/application-roles.properties

Adatbázis schema

Lottery project

```sql
CREATE TABLE event (
  event_id SERIAL NOT NULL,
  event_puller CHARACTER VARYING(100) NOT NULL,
  event_prizepool INTEGER NOT NULL,
  event_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  CONSTRAINT PK_EVENT_ID PRIMARY KEY (event_id)
);

ALTER TABLE event OWNER TO postgres;

CREATE TABLE drawnnumber (
  drawnnumber_id SERIAL NOT NULL,
  drawnnumber_event_id INTEGER NOT NULL,
  drawnnumber_value INTEGER NOT NULL,
  CONSTRAINT PK_DRAWNNUMBER_ID PRIMARY KEY (drawnnumber_id),
  CONSTRAINT FK_DRAWNNUMBER_EVENT FOREIGN KEY (drawnnumber_event_id)
    REFERENCES event (event_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT
);

ALTER TABLE drawnnumber OWNER TO postgres;
```

create-schema.sql

Aktuális és az összes sorsolás listázása

RESTful szolgáltatások

- ▷ http://localhost:8080/lottery/api/service/event/latest
  - **LotteryRestService** (lot-webservice)
    - EventStub getLatestEvent() throws AdaptorException;
  - **LotteryFacade** (lot-ejbservice)
    - EventStub getLatestEvent() throws AdaptorException;
  - **EventService** (lot-persistence)
    - Event readLatest() throws PersistenceServiceException;
    - ```
      1 SELECT e
      2 FROM Event e
      3   JOIN FETCH e.numbers
      4 ORDER BY e.date DESC
      ```
    - SetMaxResult(1)
    - **Fontos!** A kihúzott számok bekötése miatt a lekérdezés az összes eseményt lekéri az adatbázisból.
  - **EventConverter** (lot-ejbservice): EventStub to(Event event);
- ▷ http://localhost:8080/lottery/api/service/event/list
  - List<EventStub>...
  - List<Event>...
      ```

JOIN FETCH

ismétlés

```
1  SELECT
2    event0_.event_id AS event_id1_1_0_,
3    numbers1_.drawnnumber_id AS drawnnum1_0_1_,
4    event0_.event_date AS event_da2_1_0_,
5    event0_.event_prizepool AS event_pr3_1_0_,
6    event0_.event_puller AS event_pu4_1_0_,
7    numbers1_.drawnnumber_event_id AS drawnnum3_0_1_,
8    numbers1_.drawnnumber_value AS drawnnum2_0_1_,
9    numbers1_.drawnnumber_event_id AS drawnnum3_1_0__,
10   numbers1_.drawnnumber_id AS drawnnum1_0_0__
11 FROM
12   event event0_
13     INNER JOIN drawnnumber numbers1_ ON
14       event0_.event_id=numbers1_.drawnnumber_event_id
15 ORDER BY
16   event0_.event_date DESC
```

A `Set<DrawnNumber>` **LAZY** módon van bekötve az `Event` entitáshoz. A **JOIN** beköti a lekérdezésbe a táblát (ez esetben a lekérdezés azonos lesz a bemutatottal, kivéve hogy a SELECT részben nem lesznek felsorolva a `drawnumber` tábla oszlopai), de nem *attach*-olja az entitásokat (ha bejárjuk, akkor külön lekérdezésben lekéri és csatolja). A **JOIN FETCH** e plusz lekérdezés nélkül *attach*-olja is!

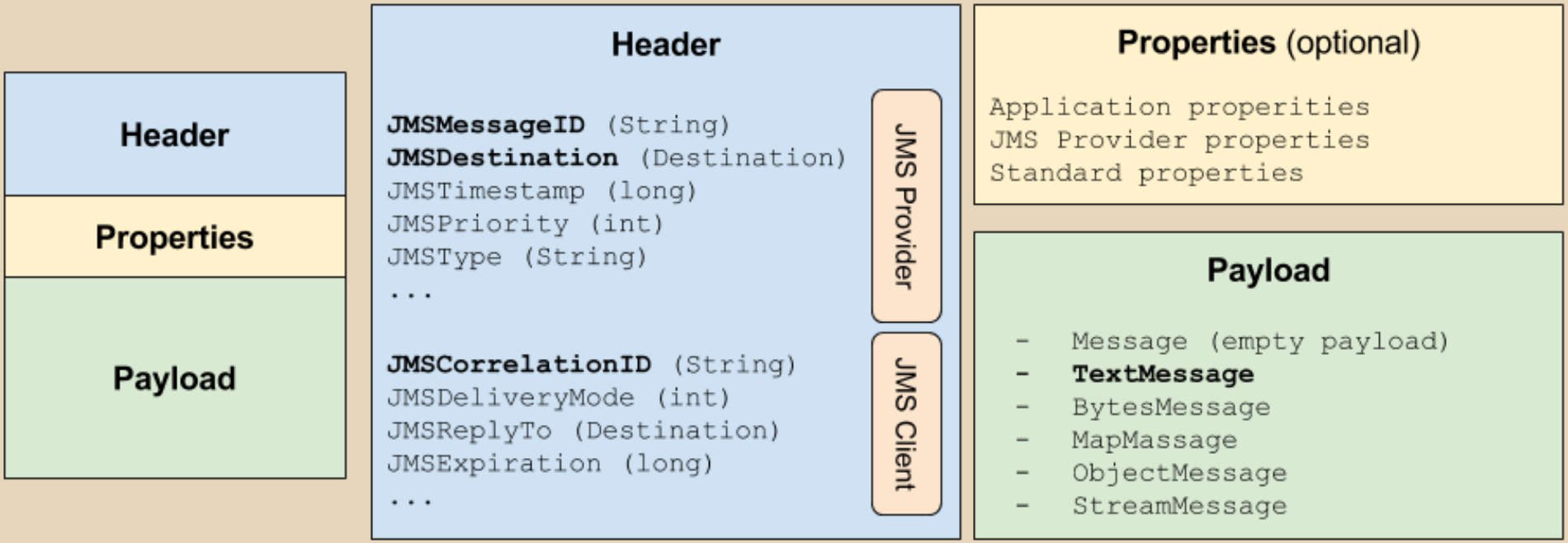
Java Message Service (JMS)

Message-Oriented Middleware (MOM)

- Üzenetküldés alapú kommunikáció
- "lazán" kötődő komponensek (beékelődik a kommunikációba az üzeneteket kezelő/tároló komponens)
- **JMS 1.1** (2002, JSR914, JEE6)
- **JMS 2.0** (2013, JSR343, JEE7)
- Típusai:
  - **point-to-point** (**queue**)
    - *producer* üzeneteket küld a queue-ba
    - *consumer* üzenetet kiolvas a queue-ból
    - egy üzenetet egy fogadó dolgoz fel (*ack* küldés is van)
    - *producer* és *consumer* nem kell hogy egy időben “online” legyen
  - **publish-subscribe** (**topic**)
    - *publisher* üzeneteket küld a topic-ba
    - *subscriber*-ek megkapják a topic-ba küldött üzenetet
    - egy üzenet több fogadó is feldolgoz(hat)
    - *publisher* és *subscriber* között van időbeli függés, "tankönyvi" eset szerint a komponensek egyszerre "online"-ok (de vannak speciális feliratkozási esetek)

- http://hornetq.jboss.org/
- Deprecated, JBoss 7.x-től **JBoss A-MQ** váltja fel
  - http://www.jboss.org/products/amq/overview/
- **JMS 1.1** és **JMS 2.0** támogatás
- Verzió: **2.3.25.Final** (JBoss 6.4 esetén)

JMS üzenet felépítése



JMS Queue létrehozása

HornetQ

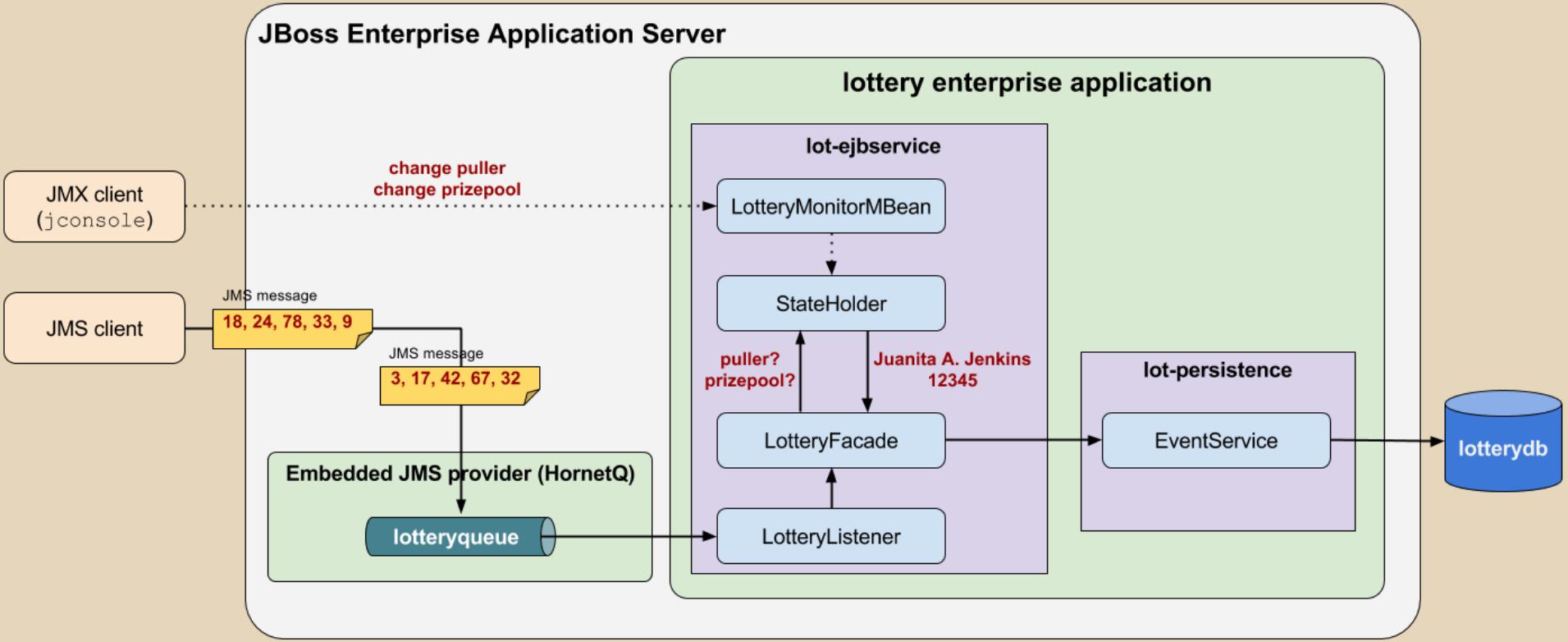
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
3    <hornetq-server>
4      <jms-destinations>
5        <jms-queue name="lotteryqueue">
6          <entry name="jms/queue/lotteryqueue" />
7          <entry name="java:jboss/exported/jms/queue/lotteryqueue"
               />
8        </jms-queue>
9      </jms-destinations>
10   </hornetq-server>
11 </messaging-deployment>
```

java:/jms/queue/lotteryqueue lesz a local JNDI név. A remote JMS client a java:jboss/exported/ előtagot automatikusan fogja használni (JBoss JMS Client jar használata esetén).

lotteryqueue-jms.xml

A file nevének *-jms.xml-nek kell lennie, és a *deployments* könyvtárba másolással "létrehozható" a destination. Másik megoldás lehet a standalone.xml-en keresztüli definiálás, illetve programozottan (*runtime*) is létrehozható queue/topic.

Új esemény létrehozása



LotteryListener

Message Driven Bean, (lot-ejbservice project)

```
1  package hu.qwaevisz.lottery.ejbservice.listener;
2  [..]
3  @MessageDriven(name = "LotteryListener", activationConfig = { //
4      @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
           "javax.jms.Queue"),
5      @ActivationConfigProperty(propertyName = "destination", propertyValue =
           "lotteryqueue"),
6      @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
           "Auto-acknowledge") })
7  public class LotteryListener implements MessageListener {
8
9    @EJB
10   private LotteryFacade facade;
11
12   @EJB
13   private MessageConverter converter;
14
15   @PostConstruct
16   public void initialize() {
17     LOGGER.info("Lottery Listener created...");
18   }
19
20   @Override
21   public void onMessage(final Message message) {
22     [..]
23   }
24 }
```

Amikor a lotteryqueue-ba üzenet érkezik, a LotteryListener MDB aktiválódik, és az onMessage() metódusa meghívásra kerül a feldolgozandó üzenettel. Ha kivételt dob a metódus, az üzenet feldolgozás rollback-elődik, és nem kerül ki a sorból!

LotteryListener.java

LotteryListener

- **queue name**: kinyerhető (hasznos, ha egy *listener* több queue-ra figyel egyidejűleg (erre van lehetőség)
- **correlation id**: tipikusan kliens hozza létre, és elküldi a JMS message-el, hogy később pl. egy async válasz során azonosítani tudja a "választ".

```
1  @Override
2  public void onMessage(final Message message) {
3    try {
4      if (LOGGER.isDebugEnabled()) {
5        final Queue destination = (Queue) message.getJMSDestination();
6        final String queueName = destination.getQueueName();
7        LOGGER.debug("New JMS message arrived into " + queueName + " queue (correlation
             id: " + message.getJMSCorrelationID() + ")");
8      }
9      if (message instanceof TextMessage) {
10       final TextMessage textMessage = (TextMessage) message;
11       String content = textMessage.getText();
12       if (LOGGER.isDebugEnabled()) {
13         LOGGER.debug("Received message: " + content);
14       }
15       this.facade.createNewEvent(this.converter.toNumbers(content));
16     } else {
17       LOGGER.error("Received message is not a TextMessage (" + message + ")");
18     }
19   } catch (final JMSException | AdaptorException | NumberFormatException e) {
20     LOGGER.error(e, e);
21   }
22 }
```

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

LotteryListener.java

RedHat JBoss vs. Oracle WebLogic

	JBoss 6.4	WebLogic 12.2.1
JavaEE version	JavaEE 6	JavaEE 6
Logging	JBoss Logging, Log4J, ..	JDK Logging, ..
RESTful	RESTEasy 2.3.10.Final	Jersey 1.18
JPA Provider	Hibernate 4.2.18.Final	EclipseLink 2.5.2
JMS Provider	HornetQ 2.3.25.Final	WebLogic JMS
JAXB	JAXB 2.2.5-redhat-9	JAXB 2.2.10 Oracle

JMS Provider beállítása - I

JMS Server és JMS Module

WebLogic Administration Console

- Services | Messaging | JMS Servers
  - New
    - name: **demoJMSserver**
    - persistence store: none
    - target: `myserver`
- Services | Messaging | JMS Modules
  - New
    - name: **demoJMSmodule**
    - location in domain: blank
    - target: `myserver`
    - Would you like to add resources to this JMS system module? → yes

JMS Provider beállítása - II

JMS Submodule és JMS Destination (queue)

WebLogic Administration Console

- ▷ Services | Messaging | JMS Modules
  - demoJMSmodule | Subdeployments | New
    - name: **demoJMSsubmodule**
    - target/server: `demoJMSserver`
    - target: `myserver`
    - Would you like to add resources to this JMS system module? → yes
- ▷ Services | Messaging | JMS Modules
  - demoJMSmodule | Configuration | New
    - tpye: **Queue**
    - name: **lotteryqueue**
    - JNDI name: **jms/queue/lotteryqueue**
    - subdeployments: `demoJMSsubmodule`
    - target/server: `demoJMSserver`

JMS Provider beállítása - III

Connection Factory

WebLogic Administration Console

- ▷ Services | Messaging | JMS Modules
  - demoJMSmodule | Configuration | New
    - tpye: **Connection Factory**
    - name: **demoConnectionFactory**
    - JNDI name: **jms/demoConnectionFactory**
    - target: `myserver`

LotteryListener

Message Driven Bean, (lot-ejbservice project)

```
1  package hu.qwaevisz.lottery.ejbservice.listener;
2  [..]
3  @MessageDriven(name = "LotteryListener", activationConfig = { //
4      @ActivationConfigProperty(propertyName = "initialContextFactory", propertyValue =
           "weblogic.jndi.WLInitialContextFactory"),
5      @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
           "javax.jms.Queue"),
6      @ActivationConfigProperty(propertyName = "destinationJndiName", propertyValue =
           "jms/queue/lotteryqueue"),
7      @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
           "Auto-acknowledge") })
8  public class LotteryListener implements MessageListener, MessageDrivenBean {
9    [..]
10 }
```

LotteryListener.java

Különbségek

- A MessageDrivenBean implementálása nem kötelező, de lehetőséget ad az MDB életciklusába való beavatkozásra (setMessageDrivenContext(..) és ejbRemove() metódusok által).
- initialContextFactory tulajdonság (megadása külső JMS provider esetén fontos)
- destinationJndiName tulajdonság használata a JBoss által elvárt destination helyett (így az érték is a JBoss *friendly name* helyett a JNDI név lesz)

JMS Client Application

`lot-jmsclient` project

Távolról JMS üzenetet küld a `lotteryqueue`-ba, melyet az elindított JBoss EAS által indított **HornetQ** mint JMS MOM fog fogadni.

Fontos!

Kommunikálni a *JMS Provider*-rel szükséges, de mivel a HornetQ egy **embedded JMS Provider**, ezért csatlakoznunk először a JBoss-hoz kell a Remote EJB-nél már megismert `InitialContext`-en keresztül, hogy elérjük a JBoss **JNDI** fáját. Külső *JMS Provider* esetén mindezek nem így érvényesek!

Ahhoz, hogy meg tudjuk szólítani ezt a szolgáltatást, az alábbiak szükségesek:

- **JBoss initial context factory**
  - szükséges erőforrás a classpath-on: `org.jboss.naming.remote.client.InitialContextFactory`
  - `compile group: 'org.jboss.as', name: 'jboss-as-jms-client-bom', version: '7.2.0.Final'`
- JBoss EAS host-ja (**localhost**) és remote portja (def.: **4447**)
  - `standalone.xml` | socket-binding-group | `remoting` socket-binding port: 4447
- **JMS Connection Factory JNDI neve** (`standalone.xml`: `jms/RemoteConnectionFactory`)
- Egy min. guest role-lal rendelkező user sikeres authentikációja (**username** és **password**)
- A cél **queue JNDI neve** (`lotteryqueue-jms.xml`: `jms/queue/lotteryqueue`)
- Ha `TextMessage` helyett pl. `ObjectMessage`-et küldünk, akkor szükség volna egy "serviceclient.j ra is, mely tartalmazná a `Serializable` DTO-kat (hasonlóan az ejb client-nél alkalmazottak szerint).

Messaging subsystem

standalone-full.xml

```
1  <subsystem xmlns="urn:jboss:domain:messaging:1.4">
2      <hornetq-server>
3          [..]
4          <connectors>[..]</connectors>
5          <acceptors>[..]</acceptors>
6          <security-settings>
7              <security-setting match="#">
8                  <permission type="send" roles="guest"/>
9                  <permission type="consume" roles="guest"/>
10                 [..]
11             </security-setting>
12         </security-settings>
13         <address-settings>[..]</address-settings>
14         <jms-connection-factories>
15             <connection-factory name="InVmConnectionFactory">[..]</connection-factory>
16             <connection-factory name="RemoteConnectionFactory">
17                 <connectors>[..]</connectors>
18                 <entries>
19                     <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
20                 </entries>
21             </connection-factory>
22             <pooled-connection-factory
                   name="hornetq-ra">[..]</pooled-
23         </jms-connection-factories>
24         <jms-destinations>[..]</jms-destinations
25     </hornetq-server>
26 </subsystem>
```

Az alapértelmezett biztonsági beállítások végett küldeni és fogadni csak a **guest** szerepkörű authentikált felhasználók tudnak. Az authentikációt a JBoss végzi.

A kapcsolatok *remote* kiépítéséhez a RemoteConnectionFactory elérése szükséges, amit JNDI-ból az itt konfigurált **jms/RemoteConnectionFactory** néven tudunk majd elérni.

standalone.xml

```java
final Properties environment = new Properties();
environment.put(Context.INITIAL_CONTEXT_FACTORY,
     "org.jboss.naming.remote.client.InitialContextFactory");
environment.put(Context.PROVIDER_URL, "remote://localhost:4447");
environment.put(Context.SECURITY_PRINCIPAL, "jmstestuser");
environment.put(Context.SECURITY_CREDENTIALS, "User#70365");
final Context context = new InitialContext(environment);
final ConnectionFactory connectionFactory = (ConnectionFactory)
     context.lookup("jms/RemoteConnectionFactory");
final Destination destination = (Destination) context.lookup("jms/queue/lotteryqueue");
final Connection connection = connectionFactory.createConnection("jmstestuser",
     "User#70365");
final Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
final MessageProducer producer = session.createProducer(destination);
connection.start();
final TextMessage textMessage = session.createTextMessage("1, 2, 3, 4, 5");
producer.send(textMessage);
connection.close();
```

SimpleClient.java

JMS Client alkalmazás

Oracle® WebLogic

```
1  final Properties environment = new Properties();
2  environment.put(Context.INITIAL_CONTEXT_FACTORY,
       "weblogic.jndi.WLInitialContextFactory");
3  environment.put(Context.PROVIDER_URL, "t3://localhost:7001");
4  environment.put(Context.SECURITY_PRINCIPAL, "weblogic");
5  environment.put(Context.SECURITY_CREDENTIALS, "AlmafA1#");
6  final Context context = new InitialContext(environment);
7  final ConnectionFactory connectionFactory = (ConnectionFactory)
       context.lookup("jms/demoConnectionFactory");
8  final Destination destination = (Destination) context.lookup("jms/queue/lotteryqueue");
9  final Connection connection = connectionFactory.createConnection("weblogic",
       "AlmafA1#");
10 final Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
11 final MessageProducer producer = session.createProducer(destination);
12 connection.start();
13 final TextMessage textMessage = session.createTextMessage("1, 2, 3, 4, 5");
14 producer.send(textMessage);
15 connection.close();
```

SimpleClient.java

Remote JMS Client

RedHat JBoss vs. Oracle WebLogic

	JBoss 6.4	**WebLogic 12.2.1**
Provider URL	remote://localhost:4447 6	t3://localhost:7001
User name	jmstestuser	weblogic
Password	User#70365	AlmafA1#
Destination JNDI	jms/queue/lotteryqueue	jms/queue/lotteryqueue
Initial context factory	org.jboss.naming.remote.client. InitialContextFactory	weblogic.jndi. WLInitialContextFactory
Connection factory JNDI	jms/RemoteConnectionFactory	jms/demoConnectionFactory
Classpath	org.jboss.as:jboss-as-jms-client-bom:7.2.0.Final	`[WL-HOME]/wlserver/server/lib/wlthint3client.jar`

Authentikáció

JBoss esetén külön *guest user*-t hoztunk létre, míg WebLogic-nál az *Admin user*-t használhatjuk. A `Connection Factory`-t JBoss esetén a `standalone-full.xml`-ből előre konfigurálva kaptuk, míg WebLogic esetén mi hoztuk létre *Admin console*-on.

http://localhost:8080/lottery/QueueServlet?message=1,2,3,4,5

```
1  [..]
2  InitialContext context = new InitialContext();
3  QueueConnectionFactory connectionFactory = (QueueConnectionFactory)
       context.lookup("ConnectionFactory");
4  QueueConnection connection = connectionFactory.createQueueConnection();
5  QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
6  Queue queue = (Queue) context.lookup("jms/queue/lotteryqueue");
7  QueueSender sender = session.createSender(queue);
8  TextMessage textMessage = session.createTextMessage();
9  connection.start();
10
11 textMessage.setText("1,2,3,4,5");
12 sender.send(textMessage);
13 sender.close();
14 session.close();
15 connection.close();
16 [..]
```

WebLogic esetén egyetlen különbség a *Connection Factory* JNDI neve: `javax.jms.QueueConnectionFactory`.
JBoss esetén az itt megadott JNDI név (`ConnectionFactory`) a `standalone-full.xml`-ből kiolvasható (`InVmConnectionFactory`).

`SendQueueServlet.java`

Üzenetet küldeni container-en belül mindig egyszerűbb, lévén `InitialContext`-et "konfiguráció nélkül" egyszerűen elkérhetjük (hiszen már inicializálva van), illetve nem csak "remote" *connection factory*-kat érünk el.

Singleton Session Bean

Singleton jelleg

Az EJB container garantálja, hogy a Singleton Session Bean-ből ugyanazt a példányt fogja minden szálon használni.

Természetesen nem arról van szó, hogy minden a SSB-t használó klienst szépen sorbaállít a container (ez bottleneck-je lenne az egész rendszernek). Vannak **READ** és **WRITE** lock-kal rendelkező metódusai (kizólag **Container-Managed Concurrency** (`CMC`) esetén használható):

- `LockType.READ`: párhuzamosan több szálon is futhat (állapot olvasás)
- `LockType.WRITE` (def.): kizárólag egy szálon futhat (állapot módosítás)

Container-Managed Concurrency (CMC) (def.)
`@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)`

Bean-Managed Concurrency (BMC)
`@ConcurrencyManagement(ConcurrencyManagementType.BEAN)`

Bean managed

Kizárólag **Singleton Session Bean**-ek esetén van értelmezve a *Bean-Managed Concurrency*, és ez esetben engedélyezett pl. a `synchronized` és a `volatile` kulcsszavak használata.

StateHolder

Számhúzó, nyereményalap és nyereményeloszlás tárolása és kezelése

```java
package hu.qwaevisz.lottery.ejbservice.holder;
[..]
@Singleton(mappedName = "ejb/lotteryState", name = "lotteryState")
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
public class LotteryStateHolderImpl implements LotteryStateHolder {
  private String puller;
  private Integer prizePool;
  private PrizeDistibution distribution;

  @PostConstruct
  public void initialize() {
    this.puller = "Juanita A. Jenkins";
    this.prizePool = 12345;
    this.distribution = new PrizeDistibution();
  }

  @Override
  @Lock(LockType.READ)
  public String getCurrentPuller() {
    return this.puller;
  }

  @Override
  @Lock(LockType.WRITE)
  public void setCurrentPuller(String name) {
    LOGGER.info("Change Puller: " + name);
    this.puller = name;
  }
  [..]
}
```

A `prizePool` tárolása és getter/setter üzleti metódusa mindezzel teljesen azonosan elkészíthető. A SSB-nek természetesen "illendő" interface-t készíteni (`LotteryStateHolder`), mely jelen esetben a `@Local` annotációt megkapja.

LotteryFacade

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.facade;
[..]
@Stateless(mappedName = "ejb/lotteryFacade")
public class LotteryFacadeImpl implements LotteryFacade {

  private static final Logger LOGGER = Logger.getLogger(LotteryFacadeImpl.class);

  @EJB
  private EventService eventService;

  @EJB
  private LotteryStateHolder stateHolder;

  @Override
  public void createNewEvent(int[] numbers) throws AdaptorException {
    try {
      String puller = this.stateHolder.getCurrentPuller();
      Integer prizePool = this.stateHolder.getCurrentPrizePool();
      this.eventService.create(puller, prizePool, numbers);
    } catch (final PersistenceServiceException e) {
      LOGGER.error(e, e);
      throw new AdaptorException(e.getLocalizedMessage());
    }
  }
  [..]
}
```

Az EventService a *persistence* rétegben egy tranzakción belül be kell hogy beszúrja az új eseményt (event), illetve a hozzá tartozó 5 új drawnumber sort.

LotteryFacadeImpl.java

EventService

lot-persistence project

Fontos! Az Event entitás Set<DrawnNumber> numbers field-je @OneToMany annotációjában a cascade értéke CascadeType.ALL vagy PERSIST legyen!

```java
package hu.qwaevisz.lottery.persistence.service;
[..]
@Stateless(mappedName = "ejb/eventService")
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
public class EventServiceImpl implements EventService {

  @PersistenceContext(unitName = "lot-persistence-unit")
  private EntityManager entityManager;

  @Override
  public void create(String puller, Integer prizePool, int[] numbers) throws
       PersistenceServiceException {
    try {
      final Event event = new Event(puller, prizePool);
      for (final int number : numbers) {
        event.addNumber(number);
      }
      this.entityManager.persist(event);
    } catch (final Exception e) {
      throw new PersistenceServiceException("Unknown error when persisting Events! " +
           e.getLocalizedMessage(), e);
    }
  }
  [..]
}
```

persist: egy új (vagy egy törlésre jelölt) entitás létrehozása, és egyben *managed* állapotba emelése // **merge**: egy *detached* (nem *managed*) entitás létrehozása (a metódus visszaadja a *managed* entitást, az átadott *detached* példányt nem bántja)

```java
public void addNumber(Integer number) {
  this.numbers.add(new DrawnNumber(number, this));
}
```

EventServiceImpl.java

- A **JMX** technológia a JavaSE része, és természetesen a JavaEE is támogatja, szerver oldali komponensek monitorozására is használható.
- **Managed Bean**-ek létrehozása szükséges hozzá (**MBean**), melyeket az *MBean server* észlel és kezel.
- JMX klienst könnyedén írhatunk, de a szabvány csatorna lévén erre legtöbbször nincsen szükség (pl. `jconsole` egy Java SE-vel szállított kliens alkalmazás).
- Az MBean-eknek követniük kell a JMX specifikációban leírt szabályokat (JMX kliensek szabvány elérése ezáltal garantált).

MBean készítés szabályai

- Ha az implementáció `Something` class, akkor az interface `SomethingMBean` kell hogy legyen.
- Az MBean-ben **műveleteket** és **attribútumokat** definiálhatunk.
- MBean attribútum (**attribute**) definiálása:
  - Read-only `A` típusú `xyz` attribútum esetén léteznie kell egy `A getXyz()` metódusnak.
  - Írható/olvasható `A` típusú `xyz` attribútum esetén létezni kell egy `A getXyz()` és egy `void setXyz( A value )` metódusnak.
  - Nem lehet egy attribútumhoz tartozó getter/setter-t másra használni, nem lehet azonos névvel overload-olt gettert készíteni, nem lehet más az összetartozó getter/setter paraméterezése és visszatérési értékének típusa.
- MBean művelet (**operation**) definiálása:
  - Minden olyan metódus, mely nem szabványos accessor (getter) illetve mutator (setter), automatikusan műveletnek számít (pl. `B getItem( C value )`).
- Egyszerű esetben a használható/javasolt típusok a java **primitívek**, **tömbök**, és a **String**, de létezik komplexebb típus is (pl. `TabularData`).

LotteryMonitor

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management;
[..]
public class LotteryMonitor implements LotteryMonitorMBean {
  @EJB
  private LotteryStateHolder stateHolder;

  @Override
  public String getPuller() { return this.stateHolder.getCurrentPuller(); }

  @Override
  public void setPuller(String name) { this.stateHolder.

  @Override
  public Integer getPrizePool() { [..] }

  @Override
  public void setPrizePool(Integer value) { [..] }

  @Override
  public int getDistribution(int hit) { return this.stateHolder.getDistribution(hit); }

  @Override
  public int[] getDistributions() { [..]}
  @Override
  public void setDistribution(int hit, int value) { [..] }
  public void start() throws Exception { LOGGER.info("Start Lottery MBean"); }
  public void stop() throws Exception { LOGGER.info("Stop Lottery MBean"); }
}
```

A start() és a stop() metódusok a JMX MBean életciklusa során meghívódnak. Használatuk opcionális.

LotteryMonitor.java

LotteryMonitor - I

lot-ejbservice project

```
1  package hu.qwaevisz.lottery.ejbservice.management;
2  [..]
3  public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {
4
5    public LotteryMonitor() {
6      super(LotteryMonitorMBean.class, false);
7    }
8
9    @Override
10   public ObjectName preRegister(MBeanServer server, ObjectName name) throws Exception {
11     return name;
12   }
13
14   @Override
15   public void postRegister(Boolean registrationDone) {}
16
17   @Override
18   public void preDeregister() throws Exception {}
19
20   @Override
21   public void postDeregister() {}
22
23 }
```

Az ős (StandardMBean) ctor-a az MBean interface osztályát várja!

A MBeanRegistration interface használata lehetővé teszi az életciklusba való beavatkozást.

LotteryMonitor.java

LotteryMonitor - II

lot-ejbservice project

```
1  package hu.qwaevisz.lottery.ejbservice.management;
2  [..]
3  public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {
4
5    private static final String LOTTERY_STATE_HOLDER_JNDI =
         "java:global.lottery-1.0.lot-ejbservice.lotteryState";
6
7    private LotteryStateHolder getStateHolder() {
8      LotteryStateHolder holder = null;
9      try {
10       InitialContext context = new InitialContext();
11       holder = LotteryStateHolder.class.cast( context.lookup(LOTTERY_STATE_HOLDER_JNDI)
             );
12     } catch (NamingException e) {
13       LOGGER.log(Level.SEVERE, e.getMessage(), e);
14     }
15     return holder;
16   }
17
18   @Override
19   public String getPuller() {
20     LotteryStateHolder holder = this.getStateHolder()
21     return holder != null ? holder.getCurrentPuller() : "";
22   }
23 }
```

WebLogic esetén az MBean-ek nem lesznek részei az *EJB context*-nek (JBoss esetén igen), így az `@EJB` annotáció nem használható. Minden ilyen esetben a megoldás az EJB kikérése a JNDI tree-ből.

LotteryMonitor.java

JNDI tree

A JNDI nevek böngészése

WebLogic Administration Console

- Environment | Server | Configuration
  - General tab → **VIEW JNDI Tree** link
    - *Overview tab* `Binding Name` tulajdonság értéke

Deploy-olt alkalmazások EJB-inek elérése (a JNDI fa felépítése container specifikus, bár a JavaEE szabvány megpróbálta már szabványosítani):
`java:global.lottery-1.0.lot-ejbservice.lotteryState`

- `java:global`
- `lottery-1.0` → deployolt EAR neve (a pont miatt a browser ketté szedi))
- `lot-ejbservice` → az EJB module neve az EAR-on belül
- `lotteryState` → az EJB `name` attribútuma (def: impl. osztály neve)

MBean regisztrációja

lot-ejbservice project

src | main | resources | META-INF | jboss-service.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <server xmlns="urn:jboss:service:7.0"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="urn:jboss:service:7.0 jboss-service_7_0.xsd">
5     <mbean code="hu.qwaevisz.lottery.ejbservice.management.LotteryMonitor"
           name="lottery.mbean:service=LotteryMonitorMBean"></mbean>
6 </server>
```

jboss-service.xml

`lottery.mbean` lesz a topológiában a helye (`name` attribútum), `LotteryMonitorMBean` pedig ezen belül az MBean neve (`name` attribútum). A `code` értékénél az osztályt kell megadni, mely megfelel mindenben a JMX MBean szabványoknak!
Ez egy *JBoss specifikus* állomány, neve kötelezően `jboss-service.xml` kell hogy legyen.

MBean regisztrációja

root project

Gradle esetén amit a gradle ear plugin által kezelt modul (jelen esetben ez a *root* project) src/main/application könyvtárába teszünk, az része lesz az *archive*-nak (az EAR-nak).

src | main | application | META-INF | weblogic-application.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <weblogic-application xmlns="http://xmlns.oracle.com/weblogic/weblogic-application">
3   <listener>
4     <listener-class>hu.qwaevisz.lottery.ejbservice.management.ApplicationMBeanLifeCycleList
5   </listener>
6 </weblogic-application>
```

weblogic-application.xml

WebLogic esetén az MBean regiszrációt **programozottan** kell elvégezni, és ezen programozott konfigurációs osztályt az EAR container specifikus leírójában (ez esetben ez lesz a weblogic-application.xml) kell meghivatkozni.

MBean regisztrációja

lot-ejbservice project

```
1  package hu.qwaevisz.lottery.ejbservice.management; [..]
2  public class ApplicationMBeanLifeCycleListener extends ApplicationLifecycleListener {
3      private static final String MBEAN_SERVER_JNDI = "java:comp/jmx/runtime";
4      private static final String OBJECT_PACKAGE =
           "hu.qwaevisz.lottery.ejbservice.management";
5      @Override
6      public void postStart(ApplicationLifecycleEvent evt) throws ApplicationException {
7          try {
8              InitialContext context = new InitialContext();
9              MBeanServer mbeanServer = MBeanServer.class.cast(
                   context.lookup(MBEAN_SERVER_JNDI) );
10             LotteryMonitor mbean = new LotteryMonitor();
11             ObjectName oname = this.buildObjectName();
12             mbeanServer.registerMBean(mbean, oname);
13         }
14         catch (Exception e) {
15             LOGGER.log(Level.SEVERE, e.getMessage(), e);
16         }
17     }
18     private ObjectName buildObjectName() throws MalformedObjectNameException {
19         return new ObjectName(OBJECT_PACKAGE +
               ":type="+LotteryMonitor.class.getSimpleName()+",name="+LotteryMonitor.class.ge
20     }
21     @Override
22     public void preStop(ApplicationLifecycleEvent evt) throws ApplicationException {
23         [..]
24     }
25 }
```

Az ApplicationLifecycleListener osztály miatt a [WL-HOME]/wlserver/server/lib/wls-api.jar-t el kell helyezni a *classpath*-on!

ApplicationMBeanLifeCycleListener.java

```
1 > [JRE_HOME]/bin/jconsole.[bat|sh]
```

DE: JBoss esetén a `jconsole` classpath-ához hozzá kell fűzni további osztályokat (pl. `jboss-cli-client.jar`), ezért a

```
1 > [JBOSS_HOME]/bin/jconsole.[bat|sh]
```

paranccsal indítsuk el (mely hivatkozik a `[JRE_HOME]`-ban lévőre.

A JBoss AS látszódni fog a *Local process*-ek között (de ugyanezen klienssel *Remote JVM*-hez is tudunk csatlakozni).

Megjegyzés

MAC OS-en (vagy bármilyen más rendszeren) ha a JBoss nem találja meg a `jconsole.sh` futtatásakor a `[JRE_HOME]`-ot, lefuttatva a JBoss alatti `jconsole.sh`-t a `CLASSPATH` környezeti változót beállíthatjuk az aktuális terminálban, majd ugyanebben a terminálban elindítjuk a `[JRE_HOME]` alatti *jconsole*-t.

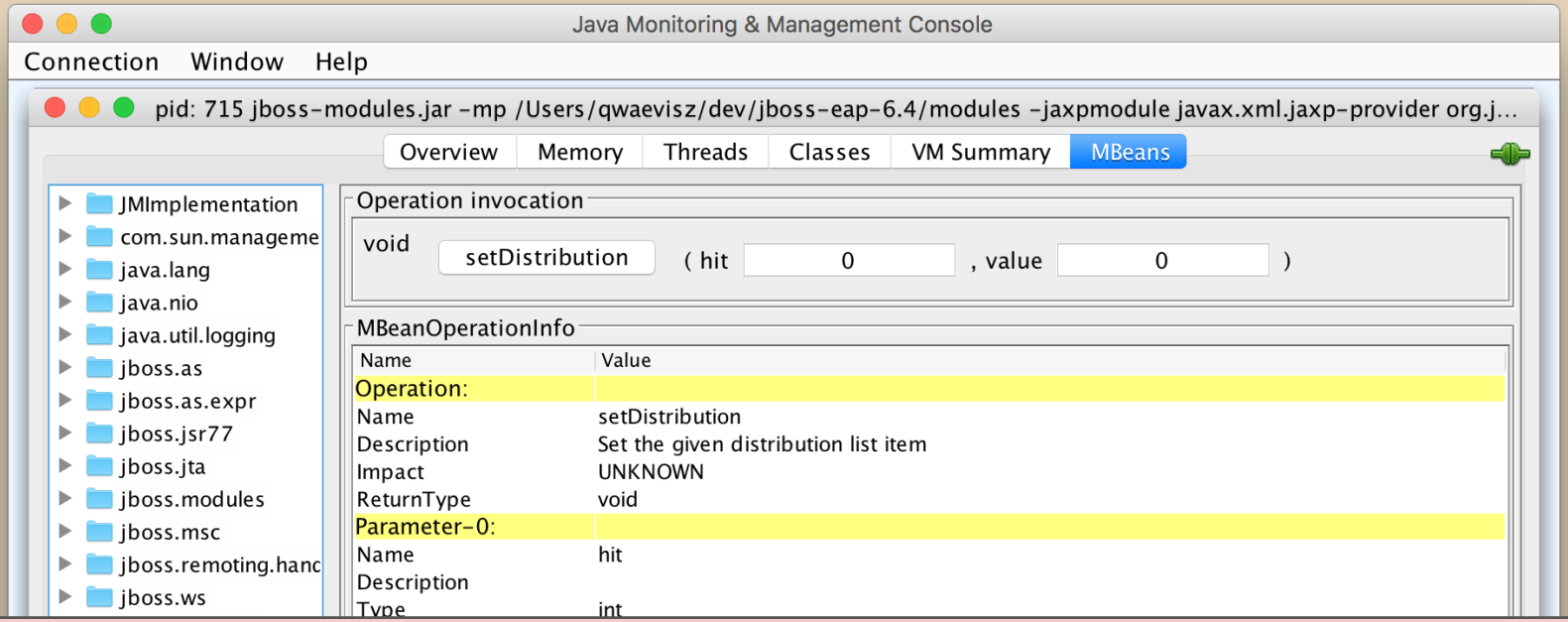
Hasonlóan a JBoss-hoz, a WebLogic esetén is szükséges a `jconsole` classpath-ához container specifikus JAR-ok hozzáadása.

```
1 > jconsole
      -J-Djava.class.path=%JAVA_HOME%\lib\jconsole.jar;%JAVA_HOME%\lib\
      -J-Djmx.remote.protocol.provider.pkgs=weblogic.management.remote
      -debug
```

A classpath beállításának egyszerű módja, ha lefuttatjuk a `setDomainEnv.[sh|cmd]` parancsot, majd ezt követően (azonos terminalban/command ablakban) a `jconsole`-t elindítjuk.

```
1 >
      [WL-HOME]/user_projects/domains/mydomain/bin/setDomainEnv.[sh|cmd
2 > jconsole
```

MBean leírások, argument nevek



Egy MBean attribútum/művelet leírása hasznos lehet, ahogy az sem volna előnytelen ha egy művelet argumentumainak forráskóban szereplő neve megjelenne a `jconsole`-ban, azonban sajnos ezen adatok átküldése kliens oldalra nem automatikus. Az MBean leszármazhat a `javax.management.StandardMBean` osztályból, melynek van számos metódusa, melyek arra szolgálnak hogy a kliens ezen adatok lekérdezésekor ezeket meghívja (egyesével). Ezen metódusok alapértelmezett viselkedését egyszerűen felülírhatjuk (*override*), ám ez még mindig egy nagyon kényelmetlen, hosszú `switch case` rengeteget jelentene, ráadásul a leírások *maintenance* feladata is rémálom volna.

Annotált MBean készítése

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management;

@Description("Lottery Monitor")
public interface LotteryMonitorMBean {

  @Description("Get current Puller")
  String getPuller();
  [..]
  @Description("Set the given distribution list item")
  void setDistribution(@PName("hit") int hit, @PName("value") int value);
}
```

LotteryMonitorMBean.java

Technika

Néhány saját annotáció bevezetésével (`@Description` és `@PName`) illetve egy köztes `AnnotatedStandardMBean` osztály készítésével (mely öröklődik a `StandardMBean`-ből és mostantól a saját MBean-jeink őse lesz) megoldhatjuk a problémát karbantartható módon is. A megoldás során **Reflection API**-t használunk.

Lottószelvény ellenőrzése

POST `http://localhost:8080/lottery/api/service/verify`

HTTP Request (application/json):

```
1 [5, 15, 20, 42, 40]
```

HTTP Response (application/json):

```
1 370
```

```
1 @Path("/service")
2 public interface LotteryRestService {
3   @POST
4   @Path("/verify")
5   @Consumes("application/json")
6   @Produces("application/json")
7   int verifyTicket(int[] numbers) throws AdaptorException;
8 }
```

LotteryRestService.java

Szelvény nyereményalapjának kiszámítása

Aktuális nyerőszámok: 10, 20, 30, 40, 50
Nyereményalap: 12345
Nyeremény eloszlás: { 1, 3, 6, 10, 80 }
Két találatos szelvény: 12345 * 0.03 = 370.35

Deploy

Alkalmazás telepítése fejlesztési környezetben

A [JBOSS_HOME] környezeti változó beállítása az alkalmazott OS függvényében:
NIX: exportJBOSS_HOME=~/dev/jboss-eap-6.4 (.bash_profile)
WIN: JBOSS_HOME=c:\apps\jboss-eap-6.4 (environment variable)

```
1  [..]
2  ext {
3    [..]
4    deployLocation = System.getenv('JBOSS_HOME') + '/standalone/deployments/'
5  }
6  [..]
7  task deployClean ( type: Delete ) {
8    delete deployLocation + "${project.name}-${version}.ear"
9    sleep(2000)
10 }
11
12 task deployEar ( type: Copy ) {
13   dependsOn 'deployClean'
14   from "build/libs/${project.name}-${version}.ear"
15   into deployLocation
16 }
```

build-root.gradle

```
1  > gradle clean build deployEar
```