JEE: Session beans & JNDI

Outline

- Stateless session beans
- How to retrieve a session bean reference
  - JNDI
  - Injection
- Configure data sources
- Development IDE

Entity vs Session

- *An **entity bean** represents domain data that can be stored into a RDB*

- *A **session bean** Accesses data, it can not be shared, it is usually read only*

Session bean

- Do not describe data, they use them instead
- Describe interactions between beans (taskflow)
  - they “link” entity beans that do not share a relation
- Implement tasks
- Execute operations not expressable in SQL

SL-SB vs SF-SB

- Stateless
  - Offer services, each service is associated to an operation (method)
  - A bean does not keep track of the states between two calls

- Stateful
  - Conversational services
  - Represent a contrinuous converstation between a client and the bean

Stateless Session Beans

- Not dedicated to a single client
  - Only a client at a time uses a single bean, but different clients can use a bean in several interactions
- No state
  - No activation, no passivation
  - No trace of what happens between different interactions
- Can access shared data

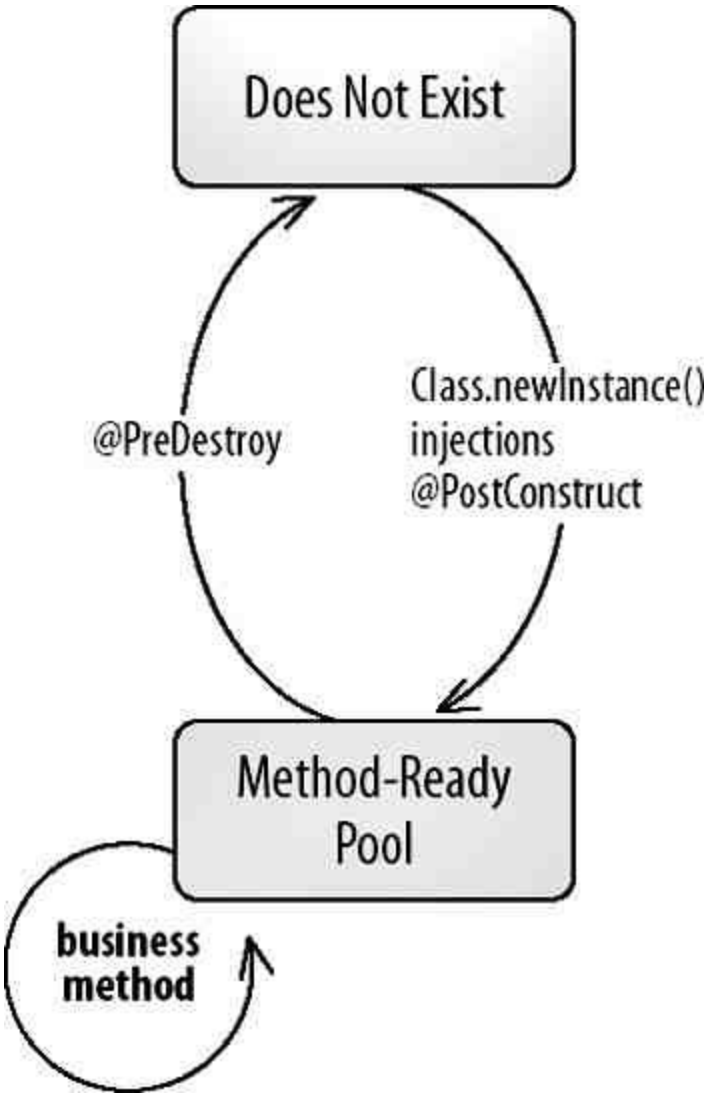
When we need a SL-SB

- Cross-cutting concerns
  - Report generation
  - Batch processes
- Some services
  - Credit cards validation
- Web Services

Life cycle

- An SL-SB life cycle is out of a programmer's control
  - No explicit instantiation
  - Has to be requested to container
- instance polling
  - Container has an instance pool
  - When a client requests an instance the container sets a reference to point to an instance in the pool
    - JNDI
    - Injection

Life cycle


Does Not Exist
Class.newInstance()
injections
@PostConstruct
@PreDestroy
Method-Ready
Pool
business
method

SL-SB Logic

- Is on the server
- clients can interact with an instance
  - POJO methods invocation
  - dot notation on the reference
- Problem: how can a client do binding if the object is on the server?

Interfaces as contracts

- An SL-SB can be contacted by a client using *Remote Interface*
- Remote interface is an interface of SL-SB that contain method that a client can invoke on the bean
- Client uses a reference to this interface
- Container invokes methods on the bean for the client

Remote interface

```
@Remote
public interface ProcessPayment
{

    public boolean byCheck(Customer customer, CheckDO check,
   double amount)
  throws PaymentException;


    public boolean byCash(Customer customer, double amount)
  throws PaymentException;


    public boolean byCredit(Customer customer, CreditCardDO
   card,
            double amount) throws PaymentException;
}
```

Note

- Call semantics is by value
  - Parameters: serialized and sent to server
  - public class Customer implements **java.io.Serializable**
- Call by reference is possible inside the EJB container only
  - It happens when a bean uses another bean
  - Both jndi lookup or Local interface (see later)

Stateless beans

- @Stateless
- ```
  public class ProcessPaymentBean
  implements ProcessPayment{
   /**Put implementation here**/
  }
  ```
- Implements the Remote interface
  - Provides an implementation for all interface methods
  ```

Stateful SB

- Similar to SL-SB:
  - Implement controller
  - Logic on server
  - Client invocation through remote interface
- Different from SL-SB
  - They do maintain a conversational status
  - An instance serves one client (until the session finishes)

Stateful SB

- @Stateful

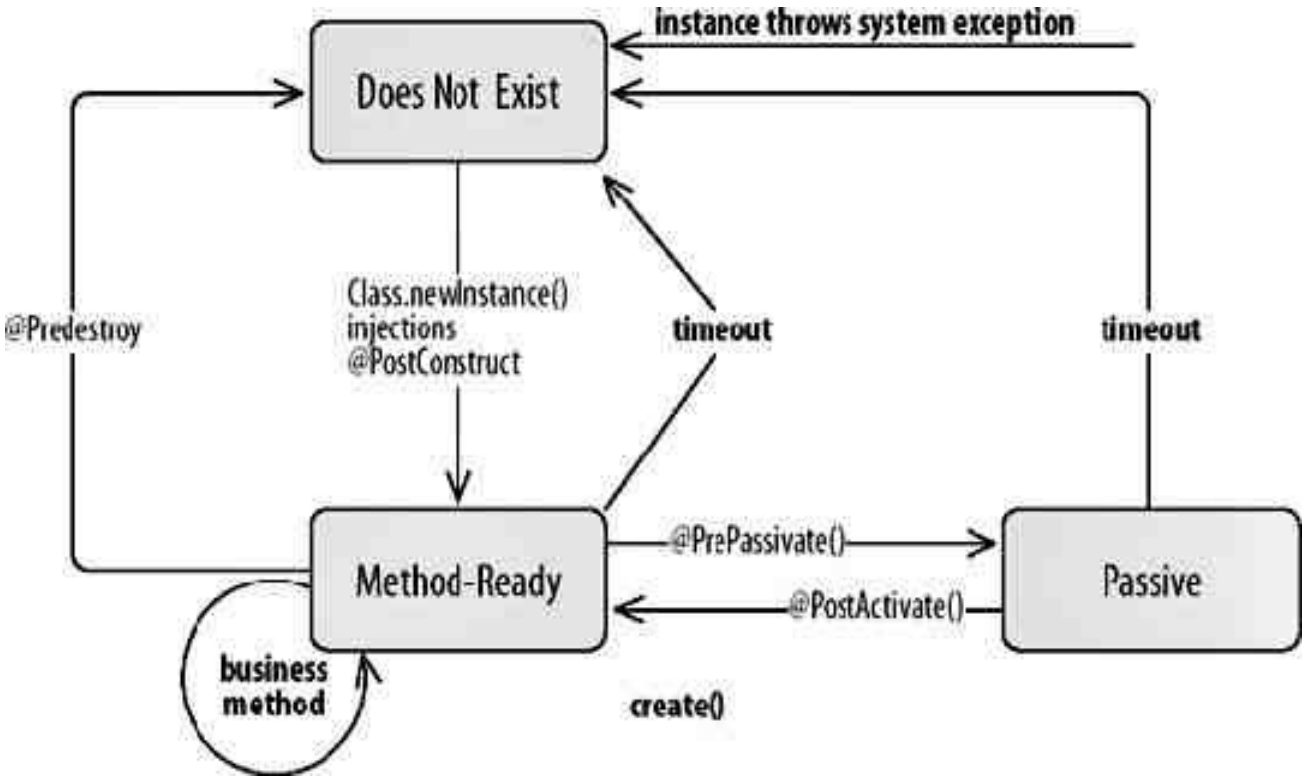
```
public class TravelAgentBean implements
  TravelAgentRemote{
   /**Put implementation here**/
  }
```

- Implements the Remote interface

Life cycle

- An SF-SB life cycle is out of a programmer's control
  - No explicit instantiation
  - Has to be requested to container
- Instances managed by the container
  - Created and destroyed as needed
  - When passivated State serialized and flushed to persistent storage
  - When activated State deserialized an attached to a newly created instance

Life cycle


instance throws system exception
Does Not Exist
@Predestroy
Class.newInstance()
injections
@PostConstruct
timeout
timeout
Method-Ready
@PrePassivate()
@PostActivate()
Passive
business
method
create()

PersistenceContext

- What happens when a method finishes to managed entities?
  - Example:

```
@PersistenceContext(unitName="titan")
   private EntityManager entityManager;
private Customer customer;
 public Customer findOrCreateCustomer(String first, String last)
{this.customer = new Customer();
                    entityManager.persist(customer)}

public void updateAddress(Address addr) {
     this.customer.setAddress(addr);//is customer managed?
}
```

Persistence context

- Customer is detached!!!
  - Persistence context does not survive method boundaries
  - Entities become detached at the end of a method
- Solution 1:
  - this.customer = entityManager.**merge**(customer);
  - The merge method updated the custromer entity on DB
- Solution 2
  - @PersistenceContext(unitName="titan", type =**EXTENDED**) private EntityManager entityManager;

Persistence context

- With extended persistence context:
  - Entities remain managed beyond method boundaries
  - If managed by a transaction commits after method ends

How to get a reference to a SB

- JNDI: lookup service
  - Portable
  - Works also outside EJB container (e.g. for clients)
- Injection: a bean is injected inside another bean (seen for entity manager before)
  - Easier
  - Only inside EJB container

JNDI lookup

Need to initialize InitialContext with Factory class and server location:

- Hashtable<String,String> env = new Hashtable<String,String>();
- env.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
- env.put(Context.PROVIDER_URL,"localhost:1099");
- InitialContext ctx = new InitialContext(env);
- ProcessPaymentRemote procpay = (ProcessPaymentRemote)ctx.lookup(“ProcessPaymentBean/remote”);

- You get a Remote interface reference

Injection

- @EJB private ProcessPaymentLocal processPayment;

- processPayment is injected with a reference to a ProcessPaymentLocal interface

Data base configuration

Data source configuration

- An enterprise application uses (usually) an external DB
- You can chose your own AS or DB
- You must have a driver
  - JDBC
  - Copy it in $JBOSS_HOME/server/default/lib
- Must define a connection pool

Connections configuration

- Configuration is made through an xml file
  - *-ds.xml
- Some properties:
  - Jndi-name: name of the resource
  - Connection-url: DB url
  - Driver-class: jdbc driver class
  - username
  - password

Riferimenti

- Burke & Monson-Haefel. *Enterprise JavaBeans 3.0.* O Reilly, fifth edition 2006.
- Ball et. al. *The Java EE 5 tutorial*. Sun Microsystems 2006. http://java.sun.com/javaee/5/docs/tutorial/doc/