Arquitectura do EJB

- Componentes EJB server side
  - **Session Beans**: parte da aplicação que faz a gestão de processos ou de tarefas. Implementam a lógica de negócio que estabelece os relacionamentos entre as entidades.
    - Use Cases correspondem (de alguma forma) a métodos dos Session Beans.
  - **Message Driven Beans**: são necessários para desenvolver a parte de coordenação de diálogo entre outros Session Beans e as entidades. Processam mensagens assíncronamente de JMS, sistemas legados ou mesmo Web Services.

- A actividade descrita num Session Bean ou num Message Driven Beam é transiente
  - Inicia-se, realiza-se e termina (é uma tarefa!)

EJB: Classes e Interfaces

- Para a criação de Session ou Message Driven beans é necessário definir:
  - A interface remota
    - Conjunto de métodos (de negócio) que podem ser acedidos por aplicações fora do container de EJB. É uma interface Java anotada com a tag `@javax.ejb.Remote`
  - A interface local
    - Define os métodos que podem ser invocados por outros beans existentes no mesmo container. Tem a anotação `@javax.ejb.Local`
  - Endpoint interface
    - Define os métodos de negócio que podem ser acedidos fora do contentor via tecnologia SOAP. Trata-se de JAVA XML-RPC e tem como objectivo ser compatível com os standard SOAP e WSDL.
    - Anotada com `@javax.ejb.WebService`

EJB: Classes e Interfaces

- Interface de Mensagem
  - Define os métodos através dos quais pode ter ligação a frameworks de messaging como o JMS

- Bean Class
  - Classe com a lógica que o bean representa
  - A classe implementa a lógica de negócio associada e tem pelo menos um dos interfaces atrás referidos.
  - O bean deve ser anotado com as tags `@javax.ejb.Stateful` ou `@javax.ejb.Stateless` consoante a sua natureza.

Beans: Exemplo Simples

- Definição de interface remota

```
import javax.ejb.Remote;

@Remote
public interface CalculatorRemote {
    public int add(int x, int y);
    public int subtract(int x, int y);
}
```

- Classe que implementa a interface

```
import javax.ejb.*;

@Stateless
public class CalculatorBean implements CalculatorRemote {
    public int add(int x, int y) {
        return x + y;
    }
    public int subtract(int x, int y) {
        return x - y;
    }
}
```

EJB Container

- Os session bean declaram interfaces que os clientes invocam
  - As aplicações clientes utilizam objectos do tipo da interface pretendida
- Os clientes fora do mesmo container invocam a interface remota
  - ou utilizam Web Services
- Clientes dentro do mesmo sistema Java EE podem utilizar a interface local, desde que estejam a correr na mesma máquina virtual.

- A arquitectura do EJB tem três componentes importantes
  - o container de beans
  - o proxy stub
  - as instâncias de bean

EJB Container

- Proxy stub:
  - quando um cliente invoca um método num session bean, não o faz directamente na instância
  - A invocação é feita ao interface remoto ou local do bean
  - Os pedidos são respondidos por um proxy stub
    - que encaminha os pedidos remotos para um contentor de beans remoto
    - encaminha as invocações da interface local para um container de beans local à virtual machine
  - Por exemplo, no caso da família de servidores JBOSS, este proxy é gerado dinamicamente em tempo de deployment. Utiliza os serviços de `java.lang.reflect.Proxy`

- EJB container
  - gere as instâncias de bean que estão contidas
  - Fornece serviços de segurança, implementação de transações, cache, etc.

EJB Container

- O container agrega a informação fornecida
  - Nas anotações existentes em cada ficheiro Java
  - Nos descritores XML

- Baseado nessa informação efectua a gestão necessária para
  - Efectuar autenticação
  - Invocar transacções
  - Gerir o ciclo de vida de um bean
  - Funcionar como mecanismo de middleware, no encaminhamento de pedidos às interfaces locais e remotas

Session e Entity Beans

- Exemplo: um TravelAgentBean cria uma reserva para um Cliente (implementa a funcionalidade expressa no Use Case)

- Aplicação cliente (excerto)

```
// Get the credit card number from the text field.
String creditCard = textField1.getText( );
int cabinID = Integer.parseInt(textField2.getText( ));
int cruiseID = Integer.parseInt(textField3.getText( ));

Customer customer = new Customer(name, address, phone);

// Create a new TravelAgent session, passing in a reference to a
// customer entity bean.
TravelAgentRemote travelAgent = ...; // Use JNDI to get a reference
travelAgent.setCustomer(customer);
// Set cabin and cruise IDs.
travelAgent.setCabinID(cabinID);
travelAgent.setCruiseID(cruiseID);

// Using the card number and price, book passage.
// This method returns a Reservation object.
Reservation res = travelAgent.bookPassage(creditCard, price);
```

Session e Entity Beans

- TravelAgent Bean
  - A implementação da funcionalidade de camada de negócio

```
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    @PersistenceContext private EntityManager entityManager;
    @EJB private ProcessPaymentRemote process;

    private Customer customer;
    private Cruise cruise;
    private Cabin cabin;

    public void setCustomer(Customer cust) {
        entityManager.create(cust);
        customer = cust;
    }
    public void setCabinID(int id) {
       cabin = entityManager.find(Cabin.class, id);
    }
    public void setCruiseID(int id) {
       cruise = entityManager.find(Cruise.class, id);
    }

    public Reservation bookPassage(String card, double price)
        throws IncompleteConversationalState {
        if (customer == null ||cruise == null ||cabin == null){
            throw new IncompleteConversationalState( );
        }
        try {

            Reservation reservation =
                new Reservation(customer,cruise,cabin,price,new Date( ));

            entityManager.persist(reservation);

            process.byCredit(customer,card,price);


            return reservation;
        }catch(Exception e){
            throw new EJBException(e);
        }
    }

}
```

Em resumo

- Beans são componentes de lógica de negócio ou entidades
- Session beans tem interfaces remotas ou locais, para efectuar serviço às aplicações cliente
  - Message driven beans e entity beans não tem interface de serviço locais ou remotas
- Três tipos de beans:
  - Entity: são persistentes e correspondem às entidades do modelo de domínio
  - Session: são pontos de contacto das aplicações cliente e implementam as tarefas decorrentes da lógica de negócio
  - Message: são pontos de integração com recurso a envio de mensagens
- O proxy stub é um construtor conceptual que gere invocações em session beans
  - As aplicações cliente não falam com os beans, mas com o proxy

EJB: serviços primários

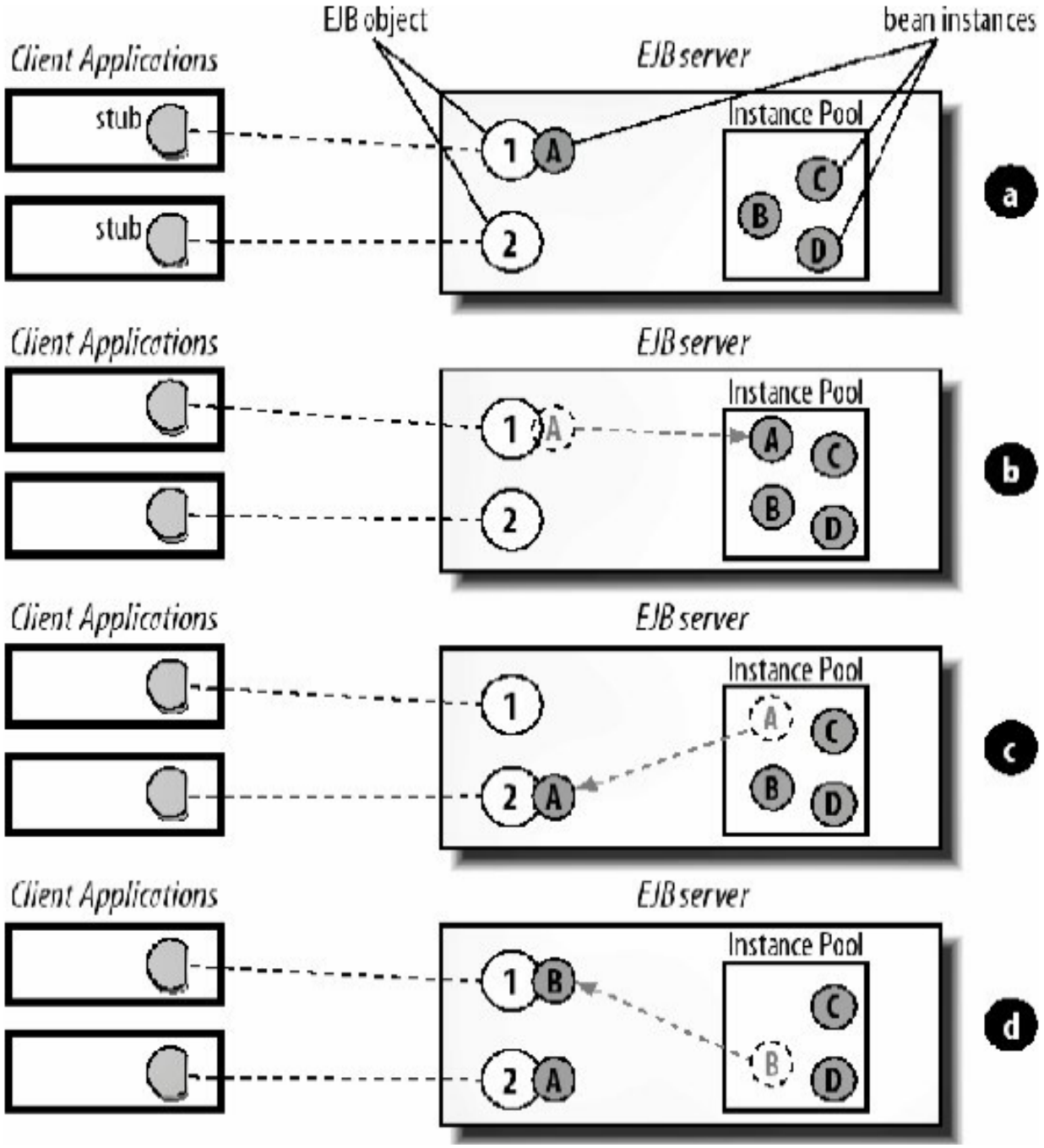
- Cenário típico de aplicações baseadas em JEE:
  - Sistema com muitos utilizadores, isto é, muitas instâncias de aplicações cliente
  - Milhares de objectos criados e em utilização
  - Muitas interacções entre os objectos (por forma a descrever a lógica de negócio)
  - Concorrência e operações com requisitos transaccionais

- O servidor de EJB tem de lidar com esta complexidade e
  - Regular, sincronizando, as interacções entre objectos
  - Partilhar recursos entre os diversos componentes
    - Exemplo típico: acesso a base de dados (pool de connections)

EJB: Serviços Primários

- Instance pooling
  - Como as aplicações clientes nunca falam directamente com os session beans, não é necessário ter um bean para cada cliente
    - Desde que a performance não se degrade os beans podem ser utilizados por mais do que uma aplicação cliente
    - No entanto um session bean é uma tarefa que não pode ser interrompida
      - possível problema: *stress de carga*
    - Especialmente útil em Stateless Session Beans na medida em que estes não guardam informação de utilização para utilização.

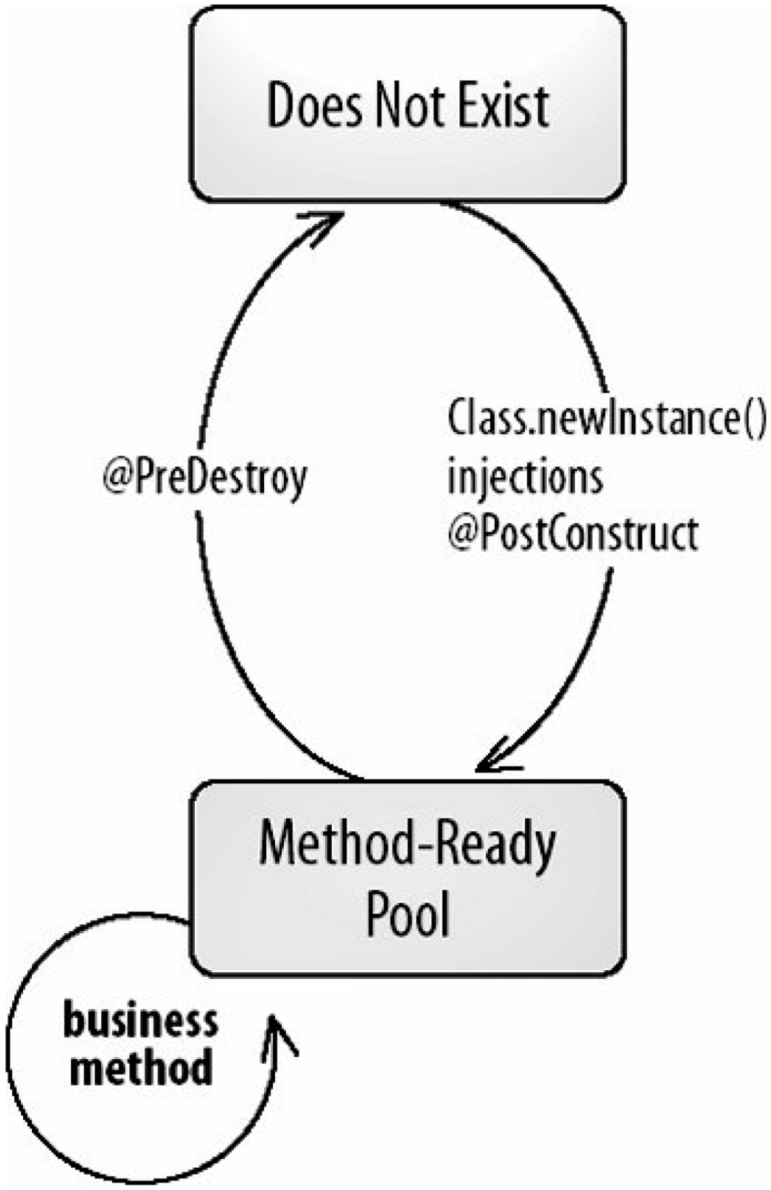
Instance Pooling



EJB: Stateful Beans

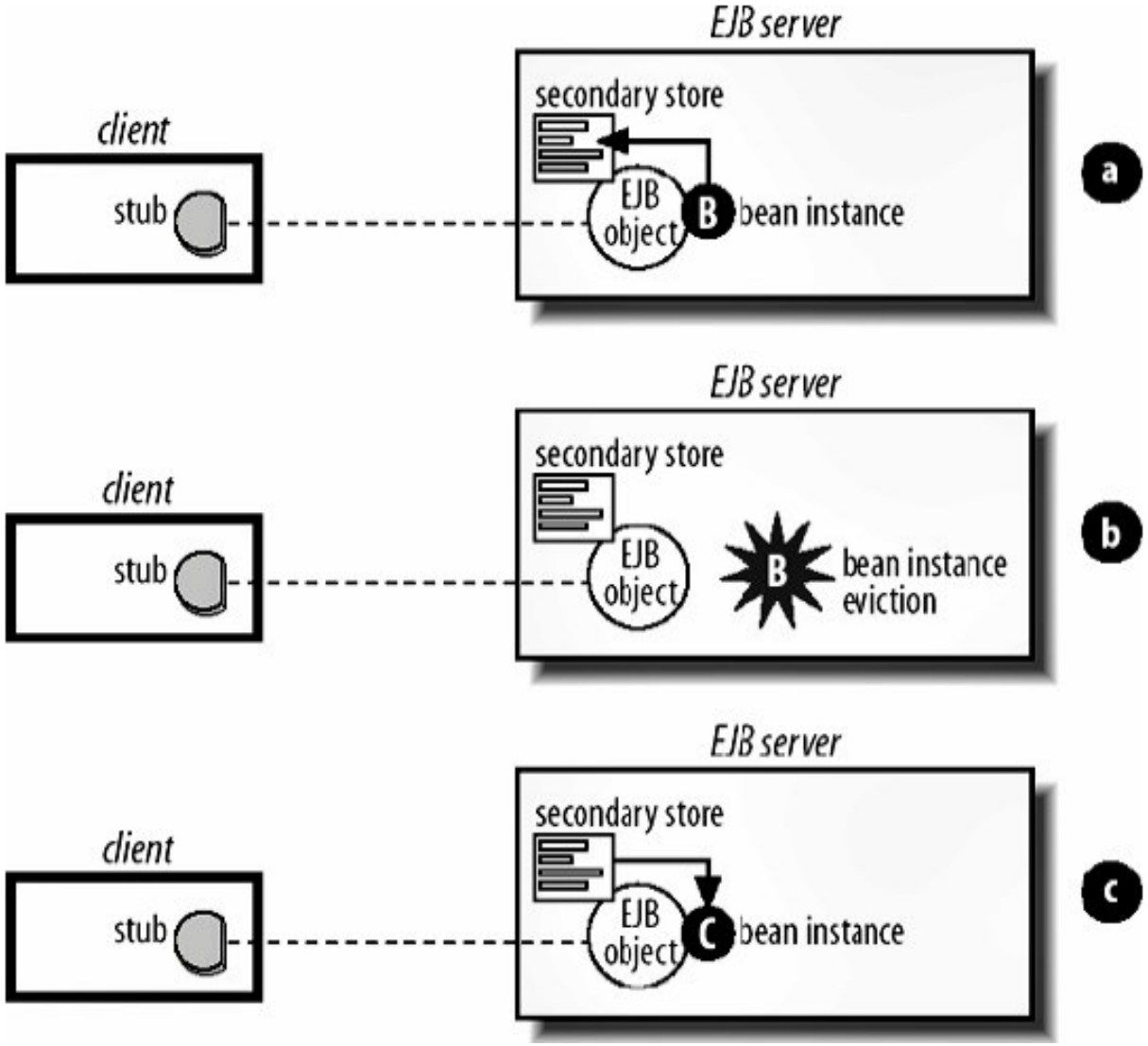
- Os Stateful Session Beans preservam o estado entre invocações de métodos
  - O bean não pode ser libertado para a pool enquanto a tarefa não terminar
- A integridade do estado de bean tem de ser preservado durante todo o diálogo com a aplicação cliente
- Stateful beans utilizam recursos do container para:
  - a sua activação
  - a sua conservação.
- Quando o bean, isto é o seu estado, é conservado, tal corresponde a serializar a informação para memória secundária.
- Quando uma aplicação cliente invoca um método no EJB, um novo objecto é instanciado e o seu estado é populado com a informação anteriormente salvaguardada.

Ciclo de Vida SessionBean Stateless

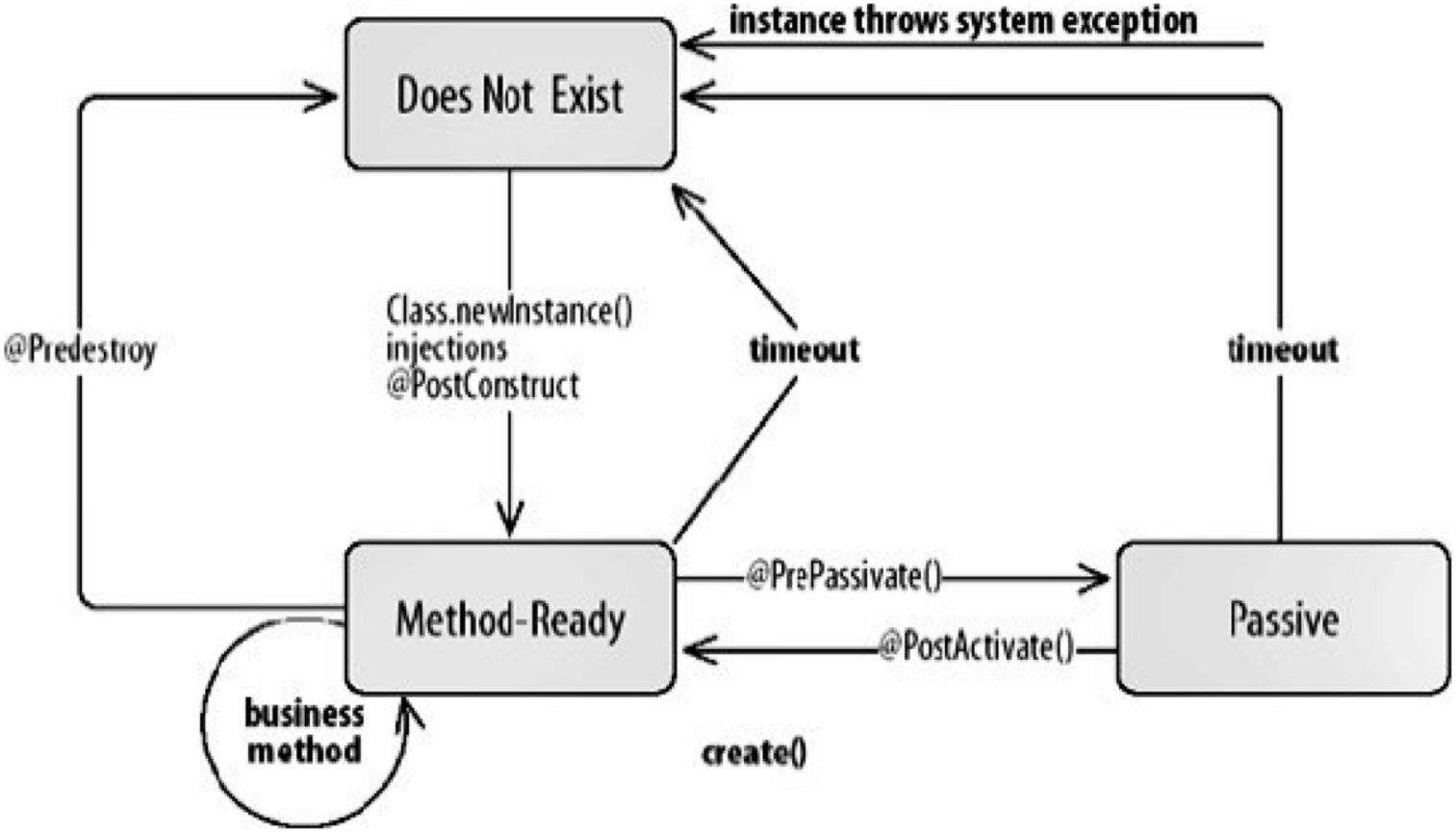


EJB: Stateful Beans – ciclo de vida

- Passivation
- Desfaz-se a associação
- Activação



Ciclo de Vida Stateful Session Bean



EBJ: Serviços Primários - Concorrência

- Concorrência com Session e Entity Beans
  - Session beans não são entidades concorrentes
    - Os Stateful beans não podem ser partilhados porque precisam de manter o estado da tarefa
    - Os Stateless beans não precisam de ser concorrentes porque não guardam nada, apenas sendo componentes de serviço
  - Os servidores EJB gerem a concorrência, pelo que os beans não precisam de ser *thread-safe*
    - Em EJB não existe a expressão `synchronized`
    - A definição dos EJB não permite que os beans possam criar threads
  - Os Entity beans podem ser acedidos concorrentemente.
    - De forma a preservar a integridade da informação a camada de persistência (Java Persistence API ou Hibernate) deve garantir este requisito.

EJB: Serviços Primários - Naming

- Um serviço de naming faz uma associação entre a referência e o objecto
  - Binding: associação de objecto remoto a um nome
  - Lookup: ligação a um serviço de directoria para pedir uma referência para um objecto

- EJB utiliza o JNDI como a API de lookup

```
javax.naming.Context jndiContext = new javax.naming.InitialContext( );


Object ref = jndiContext.lookup("TravelAgentRemote");
TravelAgentRemote agent = (TravelAgentRemote)
    PortableRemoteObject.narrow(ref, TravelAgentRemote.class);

Reservation res = agent.bookPassage(...);
```

JNDI – serviço de naming

- A configuração no ficheiro jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
 "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">

<jboss>
  ...
  <enterprise-beans>
    ...
    <session>
      <ejb-name>InventoryFacade</ejb-name>
      <jndi-name>InventoryFacadeRemote</jndi-name>
      <local-jndi-name>InventoryFacadeLocal</local-jndi-name>

      <resource-ref>
        <res-ref-name>hibernate/SessionFactory</res-ref-name>
        <jndi-name>java:/hibernate/SessionFactory</jndi-name>
      </resource-ref>

      <method-attributes>
      </method-attributes>
    </session>
    ...
  </enterprise-beans>
  ...
  <assembly-descriptor>
  </assembly-descriptor>
  ...
  <resource-managers>
  </resource-managers>

</jboss>
```

EJB: Serviços Primários - Interoperabilidade

- Suporte para RMI-IIOP (internet inter-orb protocol) e para JAX-RPC, logo para SOAP e WSDL.

- SOAP é o protocolo principal utilizado pelos Web Services.
  - Baseado em XML
  - Suporte extenso e muitas bibliotecas existentes

- Um documento WSDL é um ficheiro XML que descreve
  - Os web services disponíveis
  - Os protocolos
  - O formato das mensagens
  - Os endereços onde o serviço está disponível

Exemplo

- Entity Bean

```
import javax.persistence.*;

@Entity
@Table(name="CABIN")
public class Cabin implements java.io.Serializable{
    private int id;
    private String name;
    private int deckLevel;
    private int shipId;
    private int bedCount;

    @Id
    @Column(name="ID")
    public int getId( ) { return id; }
    public void setId(int pk) { id = pk; }

    @Column(name="NAME")
    public String getName( ) { return name; }
    public void setName(String str) {name = str; }

    @Column(name="DECK_LEVEL")
    public int getDeckLevel( ) { return deckLevel; }
    public void setDeckLevel(int level) { deckLevel = level; }

    @Column(name="SHIP_ID")
    public int getShipId( ) { return shipId; }
    public void setShipId(int sid) { shipId = sid; }

    @Column(name="BED_COUNT")
    public int getBedCount( ) { return bedCount; }
    public void setBedCount(int bed) { bedCount = bed; }

}
```

Exemplo

- Session Bean
  - Interage com a entidade Cabin

```
package com.titan.travelagent;

import javax.ejb.Remote;
import com.titan.domain.Cabin;

@Remote
public interface TravelAgentRemote {

    public void createCabin(Cabin cabin);
    public Cabin findCabin(int id);
}
```

  - Tem disponíveis os métodos `createCabin` e `findCabin`

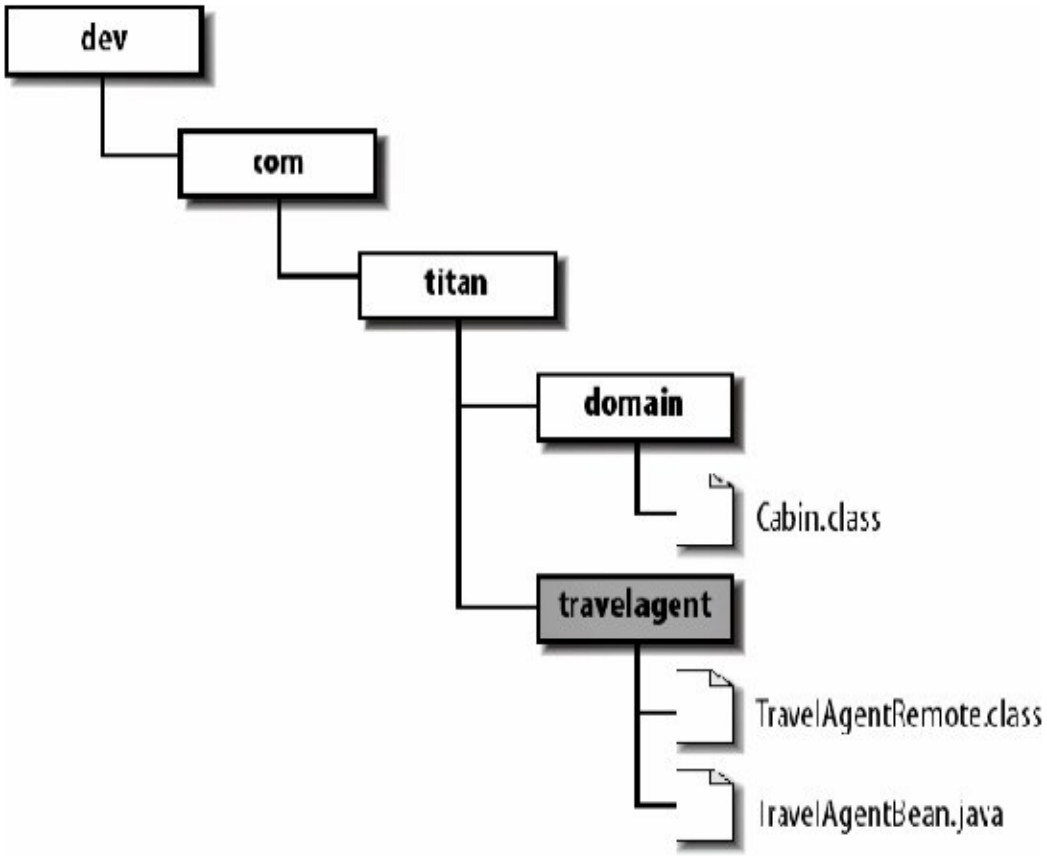
Exemplo

- A implementação do Bean TravelAgentBean

```
package com.titan.travelagent;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import com.titan.domain.Cabin;

@Stateless
public class TravelAgentBean implements TravelAgentRemote{
    @PersistenceContext
(unitName="titan")
private EntityManager manager;

    public void createCabin(Cabin cabin) {
        manager.persist(cabin);
    }

    public Cabin findCabin(int pKey) {
       return manager.find(Cabin.class, pKey);
    }
}
```

Exemplo

- Estrutura de directorias



Aplicação Cliente

```
import com.titan.travelagent.TravelAgentRemote;
import com.titan.domain.Cabin;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import java.util.Properties;
import javax.rmi.PortableRemoteObject;

public class Client {
    public static void main(String [] args) {
        try {
            Context jndiContext = getInitialContext( );
            Object ref = jndiContext.lookup("TravelAgentBean/remote");
            TravelAgentRemote dao = (TravelAgentRemote)
                PortableRemoteObject.narrow(ref,TravelAgentRemote.class);

            Cabin cabin_1 = new Cabin( );
            cabin_1.setId(1);
            cabin_1.setName("Master Suite");
            cabin_1.setDeckLevel(1);
            cabin_1.setShipId(1);
            cabin_1.setBedCount(3);

            dao.createCabin(cabin_1);

            Cabin cabin_2 = dao.findCabin(1);
            System.out.println(cabin_2.getName( ));
            System.out.println(cabin_2.getDeckLevel( ));
            System.out.println(cabin_2.getShipId( ));
            System.out.println(cabin_2.getBedCount( ));

        } catch (javax.naming.NamingException ne){ne.printStackTrace( );}
    }

    public static Context getInitialContext( )
        throws javax.naming.NamingException {

        Properties p = new Properties( );
        // ... Specify the JNDI properties specific to the vendor.
        return new javax.naming.InitialContext(p);
    }
}
```