

Stanowe komponenty sesyjne

1. Porównanie komponentów stanowych i bezstanowych.
2. Cykl życia stanowego komponentu sesyjnego,
3. Komponenty sesyjne a kontekst utrwalania,
4. Zagnieżdżanie komponentów sesyjnych,
5. Przykład: **TravelEntityBean**.

Stanowe komponenty sesyjne

Egzemplarz stanowego komponentu sesyjnego jest związany z pojedynczym klientem. Atrybuty komponentu - jego stan - są przechowywane pomiędzy kolejnymi wywołaniami metod. Pomimo tego stanowe komponenty te nie podlegają utrwalaniu.

Stanowe komponenty sesyjne pozwalają na przeniesienie części logiki biznesowej wraz ze stanem konwersacji z aplikacji klienckiej na serwer. Komponenty te działają często jako agenci, poprzez których aplikacje klienckie otrzymują dostęp do zasobów i funkcji realizowanych przez kontener EJB.

Porównanie komponentów bezstanowych i stanowych

```
package pl.edu.uj.fais.wzorce_07.beans;  
  
import javax.ejb.Remote;  
  
@Remote  
public interface ExampleRemote {  
  
    public String getResponse();  
  
}
```

Komponent bezstanowy

```
package pl.edu.uj.fais.wzorce_07.beans;

import javax.ejb.Stateless;

@Stateless
public class ExampleStateless implements ExampleRemote{
    private static int count = 0;

    private String name;
    private int number = 0;

    public ExampleStateless(){
        this.name = "Stateless_" + String.valueOf(++count);
    }

    public String getResponse() {
        return this.name + ": " + String.valueOf(++this.number);
    }
}
```

Komponent stanowy

```
package pl.edu.uj.fais.wzorce_07.beans;

import javax.ejb.Stateful;

@Stateful
public class ExampleStateful implements ExampleRemote{
    private static int count = 0;

    private String name;
    private int number = 0;

    public ExampleStateful(){
        this.name = "Stateful_" + String.valueOf(++count);
    }

    public String getResponse() {
        return this.name + ": " + String.valueOf(++this.number);
    }
}
```

Klient

```
package pl.edu.uj.fais.wzorce_07.clients;

import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import pl.edu.uj.fais.wzorce_07.beans.ExampleRemote;

public class ExampleClient implements Runnable {

    public static void main(String[] args) throws Exception {
        Properties p = new Properties();
        p.load(ExampleClient.class.getResourceAsStream(
            "jndi.properties"));
        System.getProperties().putAll(p);
        ExampleClient cl;
        Thread t;
        for (int i = 0; i < args.length; i++) {
            t = new Thread(new ExampleClient(args[i]));
            t.start();
        }
    }
}
```

Klient

```
private String sName;

public ExampleClient(String s) {
    this.sName = s;
}

public void run() {
    ExampleRemote statefulBean, statelessBean;

    try {
        Context cx = new InitialContext();
        statelessBean = (ExampleRemote)
            cx.lookup("ExampleStateless/remote");
        statefulBean = (ExampleRemote)
            cx.lookup("ExampleStateful/remote");
    } catch (NamingException ex) {
        ex.printStackTrace();
        return;
    }
}
```

Klient

```
for (int i = 0; i < 3; i++) {
    System.out.println(this.sName + ": " +
                        statelessBean.getResponse());
}

for (int i = 0; i < 3; i++) {
    System.out.println(this.sName + ": " +
                        statefulBean.getResponse());
}
}
```

Wynik działania

czwarty: Stateles_1: 1
czwarty: Stateles_1: 2
czwarty: Stateles_1: 3
czwarty: Stateful_1: 1
czwarty: Stateful_1: 2
czwarty: Stateful_1: 3
trzeci: Stateles_1: 4
trzeci: Stateles_2: 1
trzeci: Stateles_1: 5
trzeci: Stateful_2: 1
trzeci: Stateful_2: 2
trzeci: Stateful_2: 3
drugi: Stateles_1: 6
pierwszy: Stateles_3: 1
drugi: Stateles_2: 2

piaty: Stateles_1: 7
piaty: Stateles_3: 2
pierwszy: Stateles_5: 1
drugi: Stateles_4: 1
drugi: Stateful_3: 1
piaty: Stateles_2: 3
pierwszy: Stateles_1: 8
pierwszy: Stateful_4: 1
drugi: Stateful_3: 2
piaty: Stateful_5: 1
piaty: Stateful_5: 2
pierwszy: Stateful_4: 2
drugi: Stateful_3: 3
pierwszy: Stateful_4: 3
piaty: Stateful_5: 3

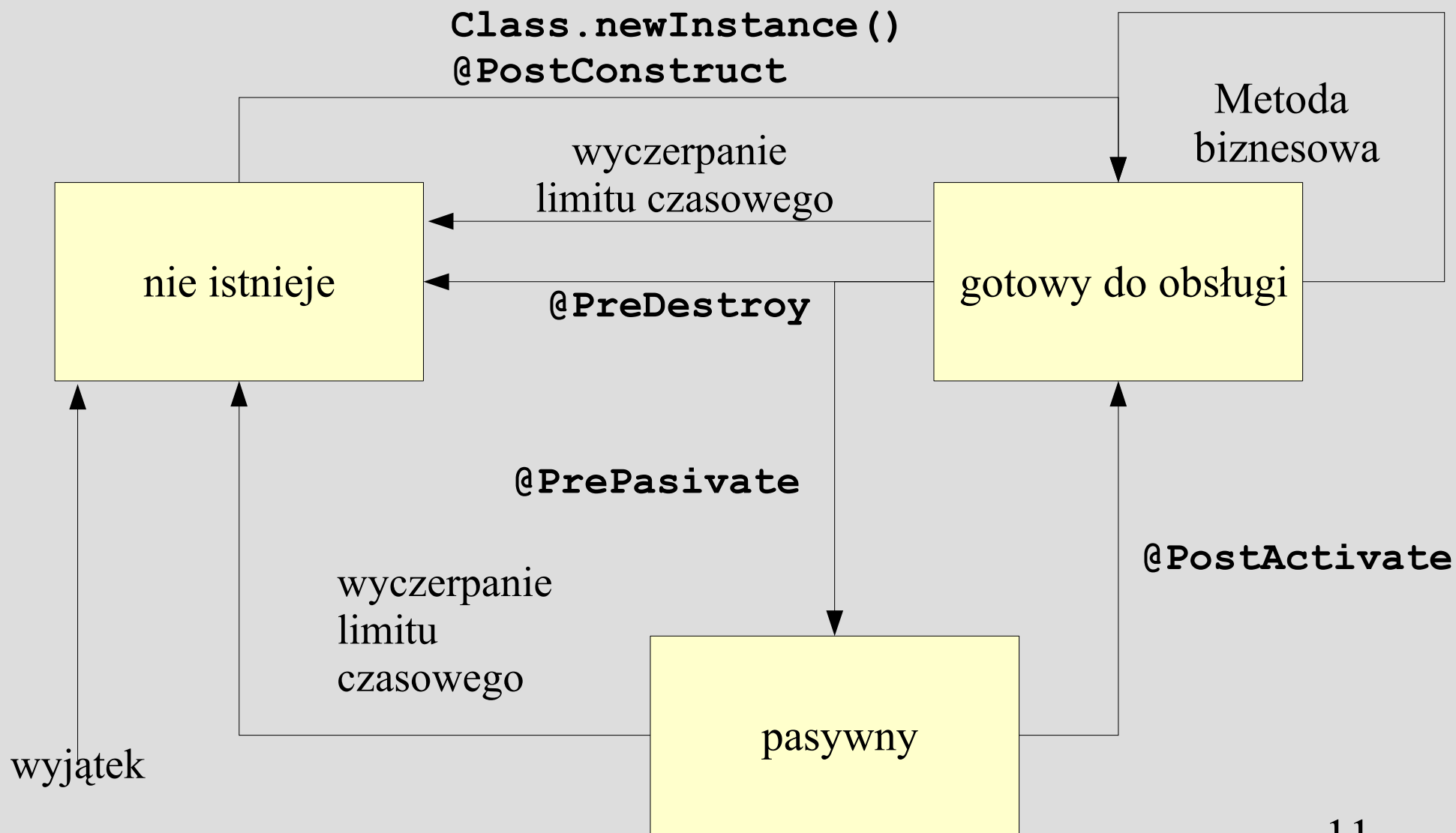
Cykl życia stanowego komponentu sesyjnego

Egzemplarze stanowego komponentu sesyjnego są dedykowane konkretnemu klientowi – niemożliwa jest wymiana i składowanie komponentów w puli.

Nie używane komponenty sesyjne są usuwane z pamięci. Aby nie stracić stanu konwersacji taki komponent jest uprzednio *pasywowany*. Cykl życia stanowego komponentu sesyjnego obejmuje trzy stany:

- nie istnieje,
- gotowy do obsługi zgłoszeń,
- pasywowany.

Cykl życia stanowego komponentu sesyjnego



Wychodzenie ze stanu „gotowy do obsługi”

Egzemplarz może przejść zarówno do stanu „nie istnieje” jak i „pasywny”.

Pasywowanie i aktywowanie komponentu może być wykonywane **dowolną** liczbę razy w trakcie cyklu życia.

Kontener wyprowadza komponent po wyczerpaniu limitów czasowych, które można określić w trakcie wdrażania poprzez mechanizmy określone przez producenta kontenera. Aplikacja kliencka może wymusić usunięcie obiektu wywołując metodę biznesową oznaczoną adnotacją **@Remove**.

```
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    ...
    @Remove
    public TicketDO bookPassage(...) {...}
}
```

Interfejs SessionSynchronization

Stanowy komponent sesyjny może obsługiwać zdarzenia związane ze zmianami stanów także poprzez interfejs `javax.ejb.SessionSynchronization`:

- `void afterBegin()` - po rozpoczęciu transakcji,
- `void beforeCompletion()` - przed zakończeniem transakcji (*commit*),
- `void afterCompletion(boolean)` – po zakończeniu transakcji z informacją o jej zakończeniu (*commit* - **true**) lub odwołaniu (*rollback* - **false**).

Stanowe komponenty sesyjne i kontekst utrwalania

Domyślnie każda metoda składowa komponentu sesyjnego jest wykonywana w ramach transakcji inicjowanej i kończonej w zakresie metody – utrwalana encja jest odłączana od kontekstu utrwalania z chwilą zakończenia przetwarzania wywołania metody.

```
public void updateAddress(Address addr) {  
    this.customer.setAddress(addr);  
    this.customer = entityManager.merge(customer);  
}
```

Stanowe komponenty sesyjne i kontekst utrwalania

Aby zwi azać zakres transakcji z istnieniem egzemplarza stanowego komponentu sesyjnego naleŹy uŹyć rozszerzonego kontekstu transakcji.

```
import static javax.persistence.PersistenceContextType.EXTENDED;
...
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    @PersistenceContext(unitName="titan", type=EXTENDED)
    private EntityManager entityManager;

    ...

    public void updateAddress(Address addr) {
        customer.setAddress(addr);
    }

    ...
}
```

Zagnieżdżanie stanowych komponentów sesyjnych

Aby skorzystać z innych komponentów sesyjnych można użyć mechanizmu wstrzykiwania.

```
@Stateful
public class ShoppingCartBean implements ShoppingCart{
    ...
    @EJB AnotherStatefulLocal another;
    @Remove void checkout() {...}
    ...
}
```

Obiekt **another** jest tworzony w momencie utworzenia komponentu **ShoppingCartBean**. Wywołanie metody **checkout ()** powoduje usunięcie tego obiektu.

Zagnieżdżanie stanowych komponentów sesyjnych

```
@Stateful
public class ShoppingCartBean implements ShoppingCart {
    @EJB AnotherStatefulLocal another;
    @PersistenceContext(unitName="titan", type=EXTENDED)
    private EntityManager entityManager;

    @Remove void checkout() {}
    ...
}

@Stateful
public class AnotherStatefulBean implements AnotherStatefulLocal {
    @PersistenceContext(unitName="titan", type=EXTENDED)
    private EntityManager entityManager;
    ...
}
```

Pola `this.entityManager` i `another.entityManager` zawierają referencję do tego samego kontekstu utrwalania.

TravelAgentBean

```
package com.titan.travelagent;

import com.titan.processpayment.CreditCardDO;
import javax.ejb.Remote;
import com.titan.domain.Customer;
import com.titan.domain.Address;

@Remote
public interface TravelAgentRemote {

    public Customer findOrCreateCustomer(String first, String last);

    public void updateAddress(Address addr);

    public void setCruiseID(int cruise);

    public void setCabinID(int cabin);

    public TicketDO bookPassage(CreditCardDO card, double price)
        throws IncompleteConversationalState;
}
```

TravelAgentBean

```
package com.titan.travelagent;

import com.titan.processpayment.*;
import com.titan.domain.*;
import javax.ejb.*;
import javax.persistence.*;
import java.util.Date;

@Stateful
public class TravelAgentBean implements TravelAgentRemote {

    @PersistenceContext(unitName="titan")
    private EntityManager entityManager;

    @EJB private ProcessPaymentLocal processPayment;

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

TravelAgentBean

```
public Customer findOrCreateCustomer(String first, String last){
    try {
        Query q = entityManager.createQuery(
            "from Customer c where c.firstName = :first
            and c.lastName = :last");
        q.setParameter("first", first);
        q.setParameter("last", last);
        this.customer = (Customer)q.getSingleResult();
    } catch (NoResultException notFound) {
        this.customer = new Customer();
        this.customer.setFirstName(first);
        this.customer.setLastName(last);
        entityManager.persist(this.customer);
    }
    return this.customer;
}

public void updateAddress(Address addr) {
    this.customer.setAddress(addr);
    this.customer = entityManager.merge(customer);
}
```

TravelAgentBean

```
public void setCabinID(int cabinID) {
    this.cabin = entityManager.find(Cabin.class, cabinID);
    if (cabin == null) {
        throw new NoResultException("Cabin not found");
    }
}

public void setCruiseID(int cruiseID) {
    this.cruise = entityManager.find(Cruise.class, cruiseID);
    if (cruise == null) {
        throw new NoResultException("Cruise not found");
    }
}
```

TravelAgentBean

```
@Remove
public TicketDO bookPassage(CreditCardDO 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);
        processPayment.byCredit(customer, card, price);
        TicketDO ticket = new TicketDO(
            customer, cruise, cabin, price);

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

Podsumowanie

Stanowe komponenty sesyjne wypełniają lukę pomiędzy usługami realizowanymi przez serwer (komponenty bezstanowe) a aplikacjami klienckimi, pozwalając na zredukowanie ich jedynie do warstwy prezentacji.