Enterprise Java Beans 3.0

Lukáš Zapletal
liberix.cz

EJB 3.0 a JPA 1.0

JavaBean - vysvětlení pojmu

- Java třída s get/is a set metodami
- má tedy vlastnosti
- žádné další podmínky nejsou kladeny
- JavaBean je tedy třída, která má alespoň jednu vlastnost (nebo více)
- stačí pro čtení nebo pro zápis
- typicky pro čtení i zápis (get/set)
- ukázka Java Beanu

EJB - Enterprise Java Beans

- obohacené Java třídy o anotace
- s Java Beany mají společné pouze jméno
- anotace blíže popisují EJB
- pro dřívější verze - XML
- nyní konfigurace výjimkou
- XML stále lze použít (zpětná kompatibilita)

EJB - Enterprise Java Beans

- jsou to komponenty, které se skládají dohromady v aplikace
- běží na aplikačním serveru, který poskytuje dostatečné běhové prostředí a služby
- rozlišujeme tři typy EJB komponent:
  - session beans - v nich je logika aplikace
  - message-driven beans - reagují na události
  - entities - reprezentují uložená data

EJB - vlastnosti EJB 3.0

- perzistence dat
- škálovatelnost
- distribuovatelnost a transakce
- víceuživatelská bezpečnost
- deklarativní metadata *
- konfigurace výjimkou *
- přenositelnost a znovupoužitelnost

* - novinka

EJB - historie

- EJB 1.0 - první verze, session a entity beans
- EJB 1.1 - XML deployment descriptor nahradil serializované objekty
- EJB 2.0 - lokální intefrace, přibyly MDB, vylepšené entity beansy, EJB QL
- EJB 2.1 - web services, časovač
- EJB 3.0 - přepracovány entity beansy (nyní se jmenují prostě “entities”), anotace, mnoho změn pro zjednodušení vývoje

EJB 3.0 - zjednodušení

- dependency injection
  - vpíchnutí závislosti
  - opačný přístup (nyní push dříve pull)
  - dříve - centrální repozitář a JNDI
- interceptors (callback methods)
  - není třeba impl. hodně servisních metod
  - potřebné metody se označí

EJB 3.0 - zjednodušení

- perzistence pomocí POJO
  - už žádné home interfaces a ejbCreate()
  - nový objekt EntityManager starající se o život beanů
- snadná konfigurace pomocí metadat
  - už žádné XMl konfigurace
  - zpětná kompatibilita ale zajištěna
  - XML stále můžete používat

EJB - použití

- po vytvoření komponenty se zabalí do archivu a nahrají na aplikační server (deployment)
- tam se aktivují a jsou dostupné buď
  - **lokálně** (v rámci aplikačního serveru) nebo
  - **vzdáleně** (přes síť)
- EJB se poté volají například z
  - webové vrstvy (lokálně nebo vzdáleně)
  - desktopové aplikace (vzdáleně)

EJB 3.0 - nevýhody

- ačkoli se 3.0 hodně zlepšila, stále tu jsou
- křivka učení není strmá (nepíšete ihned)
- hůře se testuje (testování v kontejneru)
- složitější nasazení a spouštění aplikace
- EJB není povinnou součástí J2EE aplikace
- EJB lze snadno nahradit jinou komponentovou technologií, například Spring Framework (Spring Beans)

EJB - Session Beans

- slouží k implementaci vlastní logiky aplikace
- obvykle obsahují hlavní rutiny (které např. manipulují s databázovými objekty)
- rozeznáváme
  - Stateless Session Bean
    - neuchovává stav sezení
    - příklad - CardServiceBean (kontrola kreditních karet)
  - Stateful Session Bean
    - uchovává stav po dobu sezení
    - příklad - ShoppingCartBean (nákupní košík)

EJB - Session Beans

- u obou typu beanů musíme použít anotaci: @Stateless nebo @Stateful
- bean musí implementovat jedno (nebo obě) rozhraní označené jako @Local a @Remote
  - local - přístup lokálně
  - remote - přístup přes síť
- typicky se vytvoří jedno rozhraní s operacemi a Local a Remote rozhraní budou potomci

EJB - Session Beans

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

```
public interface Calculator
{
   int add(int x, int y);
   int subtract(int x, int y);
}
```

```
import javax.ejb.Remote;
@Remote
public interface CalculatorRemote extends Calculator
{
}
```

```
import javax.ejb.Local;
@Local
public interface CalculatorLocal extends Calculator
{
}
```

EJB - Session Beans

- stateful session bean - stejná situace, pouze jiná anotace (@Stateful)
- typické pojmenování:
  - SearchFacade - hlavní rozhraní
  - SearchFacadeLocal - lokální rozhraní
  - SearchFacadeRemote - vzdálené rozhraní
  - SearchFacadeBean - vlastní kód

EJB - Session Beans

- stav objektu lze kontrolovat v metodách, které označíme anotacemi
  - @PostConstruct
  - @PreDestroy
  - @PreActivate
  - @PrePassivate

EJB - Session Beans

- API nabízí také tzv. interceptory
- opět pomocí anotací lze snadno “přibalovat” kód k jednotlivým metodám
- příklad jednoduchého i/u v jedné třídě

```
@Stateless
public class HelloWorldBean implements HelloWorld {

  public void sayHello() {
    System.out.println("Hello!");
  }
```

```
@AroundInvoke
public Object myInterceptor(InvocationContext ctx) throws Exception {
  System.out.println("checkPermission interceptor invoked");
  if (!userIsValid(ctx. getEJBContext().getCallerPrincipal())) {
    throw new SecurityException("Caller: '" +
    ctx.getEJBContext().getCallerPrincipal().getName() +
    "' does not have permissions for method " +
    ctx.getMethod());
  }
  return ctx.proceed();
 }
}
```

EJB - Message Driven Beans

- používají se pro implementace příjemců zpráv
- odesílatele implementujeme jako standardní session beany
- pro transfer se používá JMS
- aplikační server musí mít podporu pro JMS
- pokud aplikace nepoužívá Webové Služby (WS), pak se obvykle využívá právě JMS a MDB

EJB - Message Driven Beans

- opět použijeme anotace
- v anotaci můžeme nastavit typ a adresu cíle
- JMS používá dva typy messagingu:
  - point-to-point (queues)
  - pub-sub (topics)
- a implementujeme metodu onMessage(...)

EJB - Message Driven Beans

```
import javax.ejb.MessageDriven;
import javax.ejb.ActivationConfigProperty;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig =
        {
        @ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"),
        @ActivationConfigProperty(propertyName="destination", propertyValue="queue/tutorial/example")
        })
public class ExampleMDB implements MessageListener
{
   public void onMessage(Message recvMsg)
   {
      System.out.println("----------------");
      System.out.println("Received message");
      System.out.println("----------------");
   }
}
```

EJB - Entities

- třetí typ EJB nazýváme od verze 3.0 Entity
- Entity se používají jako datové modely
- ukládají se do relačních databází (pokud chceme použít objektové, máme JDO)
- dříve Entity Beans, v nové verzi zjednodušeno
- je to POJO třída (nemusí implementovat žádná roznraní)
- anotacemi kompletně popíšeme Entitu, opět není třeba žádného XML

EJB - Entities

```
@Entity
@Table(name = "PURCHASE_ORDER")
public class Order implements java.io.Serializable
{
   private int id;
   private double total;
   private Collection<LineItem> lineItems;
   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getId()
   {
      return id;
   }
   public void setId(int id)
   {
      this.id = id;
   }
   public double getTotal()
   {
      return total;
   }
   public void setTotal(double total)
   {
      this.total = total;
   }
   public void addPurchase(String product, int quantity, double price)
   {
      if (lineItems == null) lineItems = new ArrayList<LineItem>();
      LineItem item = new LineItem();
      item.setOrder(this);
      item.setProduct(product);
      item.setQuantity(quantity);
      item.setSubtotal(quantity * price);
      lineItems.add(item);
      total += quantity * price;
   }
   @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy="order")
   public Collection<LineItem> getLineItems()
   {
      return lineItems;
   }
   public void setLineItems(Collection<LineItem> lineItems)
   {
      this.lineItems = lineItems;
   }
}
```

EJB - Entities

```
@Entity
public class LineItem implements java.io.Serializable
{
   private int id;
   private double subtotal;
   private int quantity;
   private String product;
   private Order order;
   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getId()
   {
      return id;
   }
   public void setId(int id)
   {
      this.id = id;
   }
   public double getSubtotal()
   {
      return subtotal;
   }
   public void setSubtotal(double subtotal)
   {
      this.subtotal = subtotal;
   }
   public int getQuantity()
   {
      return quantity;
   }
   public void setQuantity(int quantity)
   {
      this.quantity = quantity;
   }
   public String getProduct()
   {
      return product;
   }
   public void setProduct(String product)
   {
      this.product = product;
   }
   @ManyToOne
   @JoinColumn(name = "order_id")
   public Order getOrder()
   {
      return order;
   }
   public void setOrder(Order order)
   {
      this.order = order;
   }
}
```

EJB - Entities

- Entities používají JPA
- v EJB 3.0 je to JPA 1.0
- JPA je velmi komplexní záležitost
- implementace javax.persistence jsou momentálně dvě hlavní
  - Oracle TopLink (RI)
  - Hibernate

EJB - Entities

- nikde není nařízeno, že je nutné používat JPA
- je možné použít i jiné O/R mapovací frameworky (Cayene, Apache produkty)
- mohou se třeba použít jen session beany
- a nebo ani ty ne - další informace o programování J2EE bez EJB v další prezentaci

EJB - ukázky

- ukázka vytvoření session beanu
- klienta vytvoříme až v příští prezentaci
- ukázka JPA s ukládáním do databáze Apache Derby
- další popisy vlastností JPA již na příkladech