Großer Beleg

Vergleich Persistenzkonzepte
Newtron Framework – J2EE

Robert Vogel

Inhalt

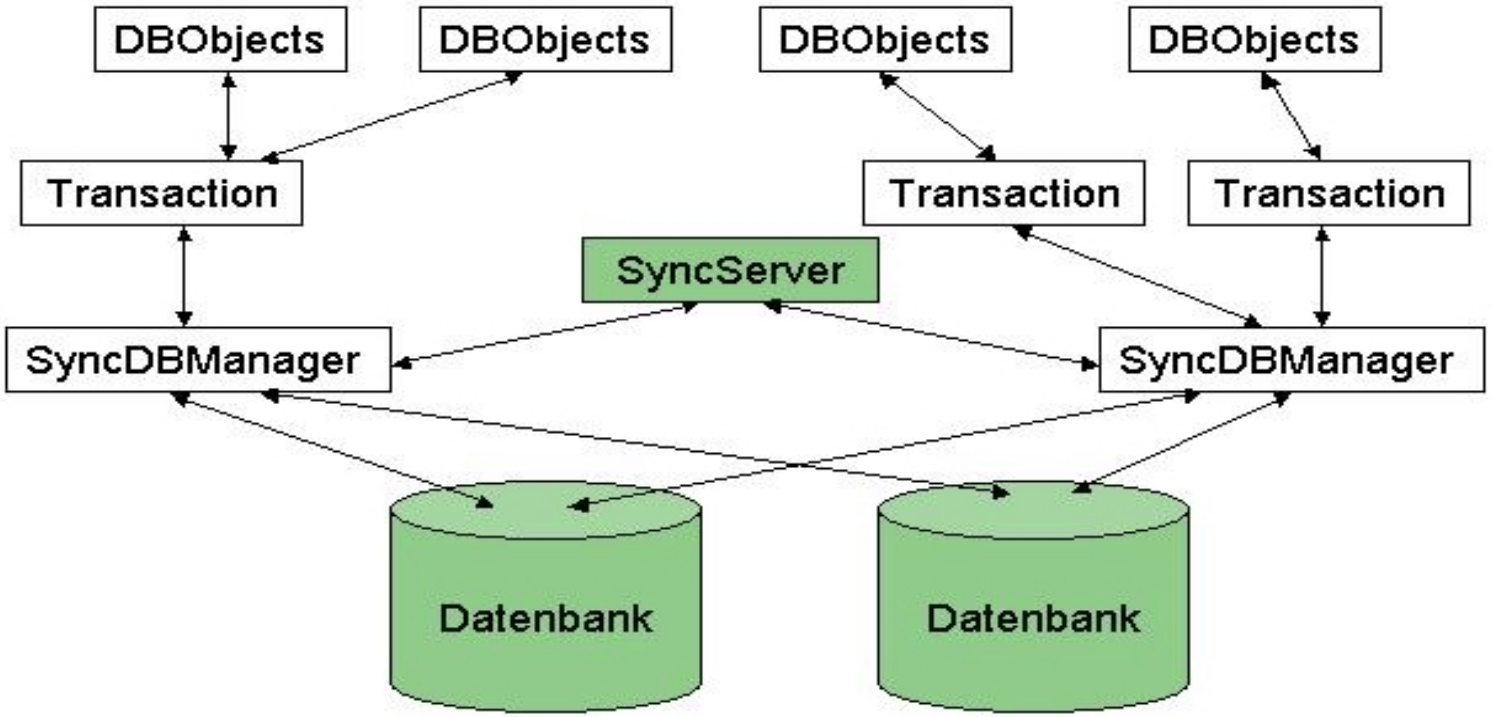
- Persistenzkonzepte Newtron
- Vergleich
- Migrationsszenarien

Persistenzkonzepte Newtron

Überblick

- objektorienter Abstraktionslayer über relationalem Datenbankschema
- JDBC basiert
- Klassen werden aus Beschreibung automatische generiert
- Features
  - Caching synchronisiert über mehrer Applikationen
  - Unterstützung von Relationen (1:1, 1:n, n:m)
  - Transaktionssteuerung, verschachtelte Transaktionen

Architektur



Syntax

- Java-Klasse von DBObject/DBIdentObject abgeleitet
- Attributbeschreibung in einem Kommentar

```
/*
   #begindescription
   addPrimaryKey("ident", "int", "");
   ...
   #enddescription
 */
```

Syntax

- `addAttribute(„value", „int", „unique number");`
  - wie addPrimaryKey, fügt aber ein Attribut hinzu
- Datentypen
  - byte, int, double, long
  - String, String50, String250, String4000
  - Date
    - java.util.Date

Funktionen von DBObject

- insert()
  - fügt ein Objekt in die Datenbank ein
- delete()
  - löscht das Objekt aus der Datenbank
- lock()
  - sperrt das Objekt optimistisch

Transaktionen

- Änderungen können nur innerhalb einer Transaktion erfolgen
- wird von DBManager erzeugt und verwaltet
- Objekte innerhalb der gleichen Transaktion sehen Änderungen sofort, außerhalb erst nach commit
- verschachtelte Transaktionen

  ```
  Transaction childTransaction = new Transaction(parentTransaction)
  ```

  - Parent-Transaktion sieht Änderungen innerhalb Child-Transaktion erst nach deren commit
  - Commit der Parent-Transaktion bewirkt kein commit der Child-Transaktion

Vergleich

Objektrelationales Mapping

Newtron

- Festlegung persistenter Attribute und Relationen erfolgt einfach im Beschreibungsteil der Klasse
- Für die direkte Manipulation von n:m-Relationen existiert eine Relationsklasse
- Mittels **contains** werden Abhängigkeiten zwischen Objekten dargestellt.

EJB

- Festlegung von Attributen und Relationen erfolgt durch abstrakte get- und set-Methoden
- Relationen werden durch den Container verwaltet
  - Relationen werden über das Local-Interface der assoziierten Bean verwaltet
  - bei n:m-Rel. keine Relationsklasse zur direkten Manipulation vorhanden
- Nur eingeschränkte Möglichkeiten Vererbung zu nutzen

Transaktionen

Newtron

- explizite Transaktionssteuerung durch Verwendung der **commit**- und **rollback**-Methoden
- Transaktionen können beliebig tief verschachtelt werden
- optimistisches Sperren durch Verwendung der **lock**-Methode

EJB

- Festlegung des Transaktionsverhaltens auf Methodenebene
  - deklarativ im Deployment Descriptor durch Transaktionsattribute
- Mechanismus für optimistisches Sperren existiert nicht
  - durch Verwendung von Versionsnummern / Zeitstempel realisiert
- Verschachtelte Transaktionen werden nicht unterstützt

Cache

Newtron

- Daten werden lokal in einem Cache gehalten
- Synchronisation der lokalen Caches erfolgt durch **SyncServer**
- bei Übergewicht lesender Zugriffe sehr effizient

EJB

- Daten werden über die Datenbank synchronisiert
  - zu Beginn einer Transaktion Daten aus DB lesen (**ejbLoad**)
  - zum Ende einer Transaktion Daten in DB schreiben (**ejbStore**)
  - auch bei lesendem Zugriff !
- im Single-Server-Betrieb kann EJB zum Cachen verwendet werden
- resultiert in einer Vielzahl von DB-Zugriffen

UUID-Generierung

Newtron

- Es existiert ein Mechanismus zur automatischen Primärschlüsselgenerierung
- Dazu muss das DB-Objekt die Klasse **DBIdentObject** erweitern und ein Attribut **ident** vom Typ **int** besitzen

EJB

- keine einheitliche Lösung
- viele Applikationsserver bietet eigenen Mechanismus
  - **nicht portabel !**
- Bsp. BEA WebLogic
  - DB-Tabelle, welche Primärschlüssel in Form eines Zählers verwaltet
  - Unterstützung SEQUENCE (Oracle)

Laufzeitumgebung

Newtron

- Eigenentwicklung der Firma Newtron
  - Wartung und Modifikation muss durch eigene Entwickler erfolgen
  - einfache Anpassung an neue Anforderungen möglich

EJB

- EJBContainer bietet eine Vielzahl von Diensten
  - Persistenz, Sicherheit, Transaktionen, Ressourcenverwaltung
- Industriestandard
  - Applikationen portabel zwischen verschiedenen App.servern
- Vielzahl von Applikationsservern verfügbar
  - auch Open-Source-Implementierungen (JBoss)
- Komponenten müssen Konventionen folgen
  - z.B. kein Einsatz von Threads
  - Asynchrone Aktivitäten durch Message-Driven Beans (seit EJB 2.0)

Toolunterstützung

Newtron

- keine Toolunterstützung vorhanden
- Codegenerierung erfolgt durch ein Shell-Skript
  - gestaltet sich relativ einfach
- weitere Tools müssten eigenständig entwickelt werden
  - da keine Standardtechnologie

EJB

- breite Palette von Tools vorhanden
- sind zur Programmierung zwingend erforderlich
  - da **Home**-, **Component**-Interface und **Implementierungsklasse** ständig synchron gehalten werden müssen

Wiederverwendbarkeit

Newtron

- Beschreibung der DB-Objekte kann wiederverwendet werden
  - dieser können neue Attribute oder Beziehungen zugefügt werden
  - durch Aufruf des Shell-Skriptes wird zugehörige Klasse generiert

EJB

- hoher Wiederverwendbarkeitsgrad
- bestehende Komponenten können zu neuen zusammengefasst werden
  - erfolgt meist durch GUI-basierte Tools
- bereits ein Vielzahl vorgefertigter Komponenten erhältlich
  - Teile einer Applikation können durch Erwerb externen Komponenten realisiert werden

Einarbeitungsaufwand

Newtron

- Funktionen sind ausführlich dokumentiert und gut verständlich
- Grundkenntnisse von JDBC erforderlich
- geringer Aufwand

EJB

- Entity Beans nur ein Teil der EJB-Technologie
  - zum Verständnis der Zusammenhänge komplette Einarbeitung notwendig (Session-, MessageDriven-Beans)
  - Für die Erstellung performanter Anwendungen Verwendung von J2EE-Patterns zu empfehlen
- Dokumentation in großem Umfang erhältlich
- großer Aufwand

Performance (Szenario)

Mittels Threads wurden 5 Clients simuliert. Jeder Client führt dabei folgende Operationen durch:

20 Logins anlegen
2 Auktionen mit
3 Bids
2 BlobHandles anlegen

in einer Schleife mit 10 Durchläufen
10x zufälliges Login aus 100 Logins anschauen
alle Auktionen und deren Gebote anschauen
ein Attachment von einer Auktion herunterladen
aller 10 Durchläufe einmal bei einer zufälligen Auktion bieten

Performance (Ergebnisse in ms)

EJB (THIN)	30020	34248	27246	28258	27407
NEWTRON (THIN)	15927	16648	16799	16468	13653
EJB (OCI)	33415	28326	34456	41538	41368
NEWTRON (OCI)	14505	14735	13944	14774	12872

Migrationsszenarien

Migrationsszenarien

1. Migration mit Quelltextkompatibilität
    a. Verwendung von Entity Bean
    b. Einsatz eines alternativen O/R-Mappings
    c. Verwendung des Newtron-Persistenzmechanismus
2. Verwendung von EJB-Technologie ohne Quelltextkompatibilität
3. Integration des Newtron-DB-Frameworks in EJB-Technologie

1a. Verwendung von Entity Beans

- Probleme bei CMP 2.0
  - Anfragen können nicht zur Laufzeit formuliert werden
  - geringer Funktionsumfang von EJB-QL
    - keine Aggregatfunktion (MIN, MAX, AVG, SUM, ORDER BY...)
    - Unterstützung von DISTINCT optional
  - Bildung von Subqueries nicht möglich
  - Rückgabewert einer select-Methode maximal ein Attribut
- Einsatz von CMP 2.0 abhängig vom Applikationsserver
  - BEA hat für alle Unzulänglichkeiten ein proprietäre Erweiterung
- EJB 2.1 beseitigt teilweise diese Mängel

1a. Verwendung von Entity Beans

- Ersetzen der DB-Objekte durch Proxies
  - ohne lokalem Cache
    - DB-Schema bestehender Anwendungen müssen auf Entity Beans abgebildet werden
    - Proxies müssen identische Funktionalität wie DB-Objekte haben
    - Entity Bean ist Remote-Objekt
    - Lesende/schreibende Zugriffe werden durch Proxy an Entity Bean delegiert
  - mit lokalem Cache
    - Aktualisierung der Proxies bei Attributänderungen
    - lesend kein Remote-Zugriff mehr erforderlich

1a. Verwendung von Entity Beans

Vorteile

- vorhandene Applikationen können ohne Quelltextänderung übernommen werden

Nachteile

- Verstoß gegen J2EE-Patterns
  - Entity Bean wird nicht durch Session Bean gekapselt
- schlechte Performance (viele Remote-Zugriffe auf Entity Bean)
- explizite Transaktionssteuerung
  - Gefahr langlaufender Transaktionen

1b. Einsatz von alternativen O/R-Mappern

- hierbei sind JDO-konforme Produkte zu bevorzugen

Vorteile

- performanter als Entity Beans
- können auch außerhalb eines App.servers eingesetzt werden
- Architektur ähnlich Newtron-DB-Framework
- integrierter Mechanismus für optimistisches Sperren
- Transaktionsbehandlung wie bei Newtron-DB-FW
- Standardtechnologie

Nachteile

- noch relative junge Technologie / nicht weit verbreitet(ausgereift)
- Synchronisation im Cluster

1c. Verwendung des Newtron-DB-FW

- JDBC–API Teil von J2EE
  - J2EE-konforme Anwendungen auch ohne Verwendung von EJB-Technologie möglich
- serverseitige Geschäftslogik wird in einem Servlet realisiert
  - Verlagerung der Geschäftslogik in ein Servlet
  - Transaktionssteuerung und Persistenz wird weiterhin durch Newtron-Framework realisiert

1c. Verwendung des Newtron-DB-FW

Vorteile

- Entwicklung einer Zwischenschicht (Proxies) entfällt
- Features des DB-Frameworks können weiterhin genutzt werden
- Einsatz von Threads, IO-Zugriffe, ... innerhalb von Servlets problemlos möglich

Nachteile

- Anschaffung eines teueren App.servers nicht gerechtfertigt
- Weiterentwicklung und Wartung des Frameworks durch eigene Entwickler notwendig
- keine Standardtechnologie (mangelnde Toolunterstützung, ..)

2. Migration ohne Quelltextkompatibilität

- Geschäftslogik innerhalb von Session Beans
  - für jedes DB-Objekte muss eine Entity Bean (CMP) existieren
  - Entity Beans werden als lokale Objekte realisiert
  - Session Bean fungiert als Session Facade (Remote-Objekt)
  - Steuerung der Transaktion durch den Container (CMT)
- optimistisches Sperren
  - wird durch Zeitstempel / Versionszähler innerhalb der Entity Bean realisiert
- Datenbankabfragen
  - falls nicht dynamisch durch **find**- oder **select**-Methoden nachgebildet
  - als Abfragesprache kommt hierbei EJB-QL zum Einsatz

2. Migration ohne Quelltextkompatibilität

Vorteile

- Einhaltung bestehender Patterns / Guidelines
- gute Eigenschaften bzgl. Performance / Wartbarkeit
- Container-Dienste werden genutzt
  - Persistenz, Transaktionen, ...
- gute Toolunterstützung

Nachteile

- großer Aufwand der Migration
- schwierig zu Automatisieren
- Einhaltung von Konventionen
  - keine Threads, IO-Zugriffe, ... innerhalb von EJBs zulässig

3. Integration in EJB-Technologie

- Entity Beans werden durch Newtron-DB-Framework ersetzt
  - Session Beans verwenden zum Zugriff auf die Persistenzschicht Newtron-DB-Framework
- Transaktionssteuerung sollte durch Container erfolgen (CMT)
  - Transaktionsgrenzen Methodenrumpf der umgebenden Session-Bean
  - DB-Objekte müssen auf die vom Container erzeugten Transaktionen zugreifen

3. Integration in EJB-Technologie

Vorteile

- Beibehaltung der gewohnten Programmierweise
- Migration bestehender Anwendungen wird erleichtert
  - innerhalb von Transaktionen befindliche Logik muss nur in eine Session Bean-Methode verschoben werden
  - Transaktionssteuerung erfolgt durch den Container (CMT)
- Performance-Vorteil gegenüber Entity Beans

Nachteile

- großer Aufwand
- DB-Objekte befinden sich im Container
  - kein lokales Cachen mehr möglich

Quellen

- Stefan Denninger, Ingo Peters. *Enterprise JavaBeans.* Addison-Wesley.
- Prof. Hußmann. *Vorlesung Softwarekomponenten* Sommersemester 2001. TU-Dresden
- Robert Vogel. *Großer Beleg - Vergleich des Persistenzkonzeptes der EJB-Technologie mit der Datenbank-Framework-Technologie von Newtron.* TU-Dresden