Enterprise JavaBeans

Исаев Айбек

Enterprise JavaBeans

Enterprise JavaBeans – это высокоуровневая, базирующаяся на использовании компонентов технология создания распределенных приложений, которая использует низкоуровневый API для управления транзакциями. EJB существенно упрощает разработку, поставку и настройку систем уровня предприятия, написанных на языку Java.

Enterprise JavaBeans

Сервер приложений j2ee состоит из двух основных элементов: **контейнер web**-приложения (JSP, JSF и т.д.) и **EJB-контейнер**. Первый служит для создания пользовательского интерфейса и слабо подходит для описания бизнес-логики приложения. Для этого используется вторая часть J2EE - EJB.

Архитектура Enterprise JavaBeans

Существует 2 основные архитектуры при разработке enterprise-приложений:

- традиционная слоистая архитектура (traditional layered architecture)
- domain-driven design (DDD)

Архитектура Enterprise JavaBeans

Традиционная слоситая архитектура предполагает разделение приложения на **4 базовых слоя**:

- слой презентации
- слой бизнесс-логики
- слой хранения данных
- слой самой базы данных.

Архитектура Enterprise JavaBeans

Архитектура **DDD** предполагает, что объекты обладают бизнесс-логикой, а не являются простой репликацией объектов БД. Многие программисты не любят наделять объекты логикой и создают отдельный слой, называемый service layer или application layer. Он похож на слой бизнесс-логики традиционной слоистой архитектуры за тем лишь отличием, что он намного *тоньще*

Enterprise JavaBeans

Отдельный EJB-компонент представляет собой компонент в том же смысле что и традиоционный JavaBeans «bean» („зерно“). Компоненты EJB выполняются внутри EJB-контейнера, который, в свою очередь, выполняется внутри EJB-сервера. Любой сервер, который в состоянии поддерживать EJB-контейнеры и предоставлять им необходимые сервисы, может быть EJB-сервером.

EJB-контейнер (The Enterprise JavaBeans container)

EJB-контейнер - это то место, где „живет'“ EJB-компонент. Как правило, в одном EJB-контейнере живет несколько однотипных EJB-компонент.

Контейнер использует системные сервисы в интересах „своих“ Компонентов и управляет их жизненным циклом.

Задачи EJB-контейнера

- Обеспечение безопасности – обеспечения защиты данных за счет предоставления доступа только для авторизованных клиентов и только к разрешеным методам.
- Обеспечение удаленных вызовов – Контейнер берет на себя все низкоуровневые вопросы обеспечения взаимодействия и организации удаленных вызовов.
-

Задачи EJB-контейнера

- Управление циклом жизни – клиент создает и уничтожает экземпляры компонентов. Тем не менее, контейнер для оптимизации ресурсов и повышения производительности системы может выполнить например: активацию и деактивизацию этих компонентов, создание их пулов и т.д.
- Управление транзакциями – обеспечивает защиту данных и гарантирует успешное подтверждение внесенных данных или откат транзакции.

EJB-компонент (The Enterprise JavaBeans component)

EJB-компонент представляет из себя Java-класс, который реализует некоторую бизнес-логику. Все остальные классы в EJB-системе либо реализуют поддержку клиент/сервер взаимодйествий между компонентами, либо реализуют некоторые сервисы для компонентов.

Существует три типа компонентов EJB:

Session, Message-driven beans и **Entity-Компоненты**

Характеристики Компонентов EJB

Компоненты EJB реализуют бизнес-логику

Разработчик создает Компонент так, как его видит клиент, и такой подход никак не зависит от вопросов взаимодействия Компонента с Контейнером и Сервером.

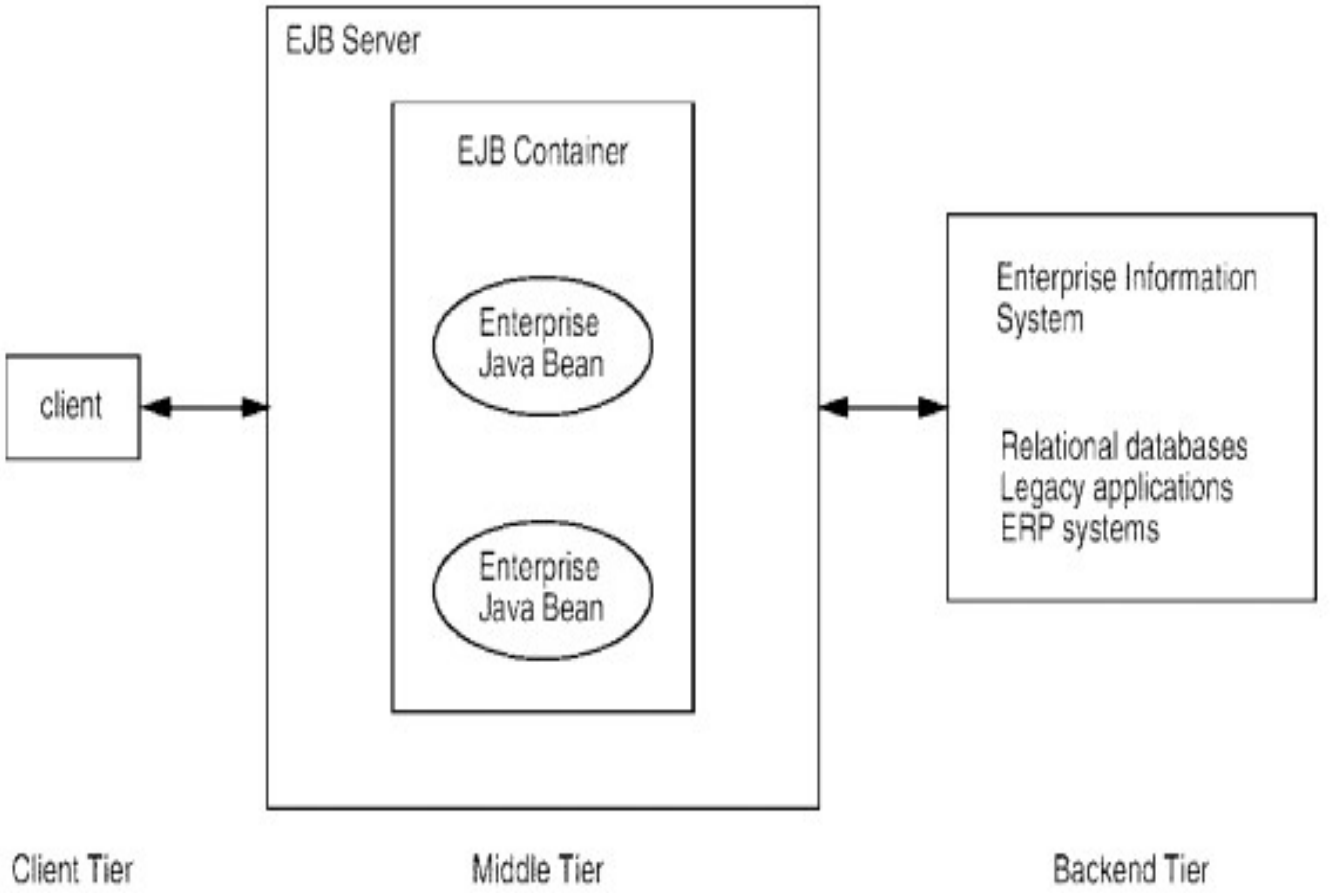
Контейнер создает экземпляры Компонентов и управляет ими во время работы приложения. Также управляет доступом клиентов к Компонентов.

Настройка Компонента осуществляется на этапе их поставки путем изменения их свойств.

Различные системные характеристики, такие, как атрибуты безопасности или транзакций, не являются частью класса Компонента.

Архитектура Enterprise JavaBeans

Рис. 2.1 Компоненты, Контейнеры и Сервера EJB.



Session beans

Session-компоненты представляет собой объект, созданный для обслуживания запросов одного клиента. В ответ на удаленный запрос клиента, Контейнер создает экземпляр Компонента.

„Представитель“ клиента на стороне EJB сервера.

Являются временными объектами, пока длится „сеанс связи“

Делятся на сохраняющих состояния(**stateful**) и нет(**stateless**).

Особенности stateless и stateful beans

Один bean может содержать множество клиентских методов. Этот момент является важным для производительности, так как контейнер помещает экземпляры **stateless**-бинов в общее хранилище и множество клиентов могут использовать один экземпляр бина.

В отличии от stateless **stateful** бины инстанцируются для каждого пользователя отдельно.

Message-driven beans

Так же как и session beans используются для **бизнесс-логики**. Отличие в том, что клиенты никогда не вызывают MDB напрямую. Обычно сервер использует MDB в **асинхронных** запросах.

Entities и Java Persistence API

Entity – компоненты представляют собой объектное представление данных из базы данных.

Состояние Entity – компонентов в общем случае нужно сохранять, и „живут “ они столько, сколько существует в базе данных те данные, которые они представляю, а не столько, сколько существуют клиентский или серверный процессы.

Entities и Java Persistence API

Одним из главным достоинством EJB3 стал новый механизм работы с persistence — возможность автоматически сохранять объекты в реляционной БД используя технологию объектно-реляционного маппинга (ORM).

Entities и Java Persistence API

В контексте EJB3 persistence провайдер - это ORM-фреймворк, который поддерживает EJB3 Java Persistence API (JPA). JPA определяет стандарт для:

- конфигурации маппинга сущностей приложения и их отображения в таблицах БД;
- EntityManager API - стандартный API для CRUD (create, read, update, delete) операций над сущностями;
- Java Persistence Query Language (JPQL) - для поиска и получения данных приложения;

Реализация

В EJB3 мы используется:

- **POJO** (Plain Old Java Objects),
- **POJI** (Plain Old Java Interfaces)
- **Аннотации.**

Аннотации

@<имя аннотации>(<список парамет-значение>)

Stateless - говорит контейнеру, что класс будет stateless session bean. Для него контейнер обеспечит безопасность потоков и менеджмент транзакций. Дополнительно, вы можете добавить другие свойства, например прозрачное управление безопасностью и перехватчики событий;

Local - относится к интерфейсу и говорит, что bean реализующий интерфейс доступен локально

Remote - относится к интерфейсу и говорит, что bean доступен через RMI (Remote Method Invocation)

EJB - применятеся в коде, где мы используем bean.

Stateful - говорит контейнеру, что класс будет stateful session bean.

и много-много других...

Условия создания session bean

В к ачестве session bean может выступать обычный класс Java удовлетворяющий следующим условиям:

- иметь как минимум один метод
- не должен быть абстрактным
- иметь конструктор по-умолчанию
- методы не должны начинаться с "ejb" (например ejbCreate, ejbDoSomething)

Интерфейсы

Интерфейс может быть помечен как :

Local - классами локальной бизнесс-логики.

Remote - обеспечит возможность работы RMI

Перехватчики

Перехватчики- объекты, методы которых вызываются **автоматически** при **вызове** метода **EJB**-бина. Объект-перехватчик является POJO, за тем лишь исключением, что метод, который должен вызываться автоматически аннотируется @AroundInvoke, например:

```
public class MyLogger {

    @AroundInvoke

    public Object logMethodEntry( InvocationContext invocationContext )
throws Exception {

          System.out.println("Entering methid: " +
invocationContext.getMethod().getName() );

          return invocationContext.proceed();

    }
}
```

Перехватчики

@Interceptors(MyLogger.class)

Пример:

```
@Interceptors( MyLogger.class )
@Stateless
public class MyClass { ... }
```