Enterprise Java Beans

EJB – это компоненты, которые предназначены для реализации так называемой бизнес-логики приложения. В отличие от веб-компонентов, которые обеспечивают веб-интерфейс приложения, EJB-компоненты не формируют какого-либо пользовательского интерфейса. Основная задача у них – обеспечить реализацию процедур обработки данных и бизнес-правил, специфичных для приложения. EJB-компоненты формируют промежуточный слой, который обеспечивает изоляцию клиентских приложений (веб-приложений, веб-сервисов, GUI-приложений) от деталей внутренней организации данных (типа и местоположения БД, используемого для связи с ними API и их внутренней структуры), предоставляя им некоторый прикладной программный интерфейс, ориентированный на решение определенного круга задач для конкретного клиента.

Как правило, система EJB-компонентов является результатом анализа и функциональной декомпозиции прикладных задач.

До появления понятия middleware и соответствующих платформ (CORBA, J2EE, MS.NET) бизнес-логику приложения приходилось «размазывать», реализуя часть ее функций в составе клиентских приложений, а часть – в составе БД средствами хранимых процедур и триггеров, что сильно затрудняло сопровождение систем и снижало их переносимость. Поскольку задачи, решаемые при разработке промежуточного слоя, различны, существуют и различные типы EJB-компонентов:

– сессионные (Session Beans);

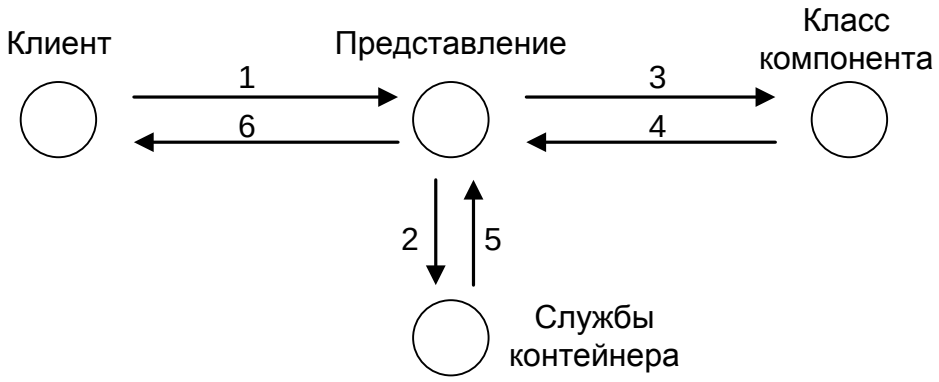
– компоненты-сущности (Entity Beans);

– компоненты, управляемые сообщениями (Message Driven Beans).

Представление компонента.

Взаимодействие клиентов с EJB-компонентами организовано не непосредственно через вызов метода класса-компонента, а через некоторый промежуточный слой, называемый представлением (view).

Представление EJB-компонентов необходимо для реализации концепции неявного промежуточного слоя (implicit middleware).



1. Клиент вызывает бизнес-метод представления.
2. Представление обращается к сервисам контейнера в зависимости от настроек, сделанных в дескрипторе на этапе разработки и/или установки.
3. Представление делегирует вызов метода классу компонента.
4. Класс компонента возвращает результат вызова метода.
5. Представление обращается к сервису контейнера.
6. Передача результата клиенту.

Реализация представления генерируется на этапе установки приложения.

С точки зрения разработчика представление компонента EJB 2.1 – это два интерфейса языка Java: домашний интерфейс (home interface) и компонентный интерфейс (component

interface). На этапе установки приложения для каждого интерфейса каждого представления контейнер генерирует класс, реализующий соответствующий интерфейс. Эти классы в совокупности с некоторыми дополнительными классами, входящими в состав сервера приложений, формируют представление компонента на этапе выполнения. Собственная функциональность компонента реализуется классом компонента.

Домашний интерфейс позволяет клиенту управлять жизненным циклом компонента. Как минимум он содержит операции создания и удаления компонента. Эти операции, реализованные в домашних классах представления компонента, предусматривают взаимодействие с инфраструктурой контейнера, которая поддерживается им для компонентов. В составе этой инфраструктуры могут быть пулы экземпляров класса компонента или класса, реализующего компонентный интерфейс. В любом случае доступ к экземпляру компонента клиент получает через домашний интерфейс, а инфраструктура контейнера от него скрыта.

Компонентный интерфейс позволяет клиенту обращаться к бизнес-методам компонента. При этом происходит взаимодействие с необходимыми сервисами контейнера и делегирование вызова классу компонента.

В сессионных компонентах EJB 3.0 вместо домашнего и компонентного интерфейсов используется единственный бизнес-интерфейс, который по назначению аналогичен компонентному интерфейсу компонентов EJB 2.1. Это стало возможным за счет отказа от явного управления жизненным циклом экземпляра компонента со стороны клиента. Так, для сессионных компонентов без состояния управление жизненным циклом компонента не актуально – для обработки вызова метода бизнес-интерфейса подходит любой экземпляр компонента. В сессионных компонентах с состоянием методы инициализации состояния экземпляра и удаления экземпляра компонента вводятся в бизнес-интерфейс.

Таким образом, сессионные компоненты и компоненты сущностей с точки зрения разработчика состоят из класса компонента, реализующего его бизнес-логику, и интерфейсов, формирующих его представление. Компоненты снабжаются XML-дескриптором, который в декларативной форме указывает, какие сервисы необходимы компоненту и как их использовать. На основе интерфейсов и дескриптора контейнер на этапе установки приложения генерирует классы, объекты которых формируют представление компонентов на этапе выполнения. Такой подход избавляет разработчика компонента от необходимости самостоятельно разрабатывать код для взаимодействия с сервисами контейнера и организации взаимодействия клиентов с компонентами. Кроме того, это позволяет менять характер взаимодействия компонента с сервисом контейнера или с клиентом без перекомпиляции класса компонента, к исходному коду которого может отсутствовать доступ на этапе установки.

В технологии EJB 3.0 у разработчика компонента появилась возможность указывать многие из аспектов взаимодействия с сервисами контейнера и клиентами при помощи аннотаций. Это позволяет значительно упростить процесс сборки приложений, где во многих случаях достаточно положиться на обеспечиваемые аннотациями значения по умолчанию вместо трудоемкого написания дескриптора. В то же время у сборщика или установщика приложения есть возможность изменить настройки, сделанные разработчиком компонента через аннотации, так как настройки в XML-дескрипторе имеют приоритет над аннотациями.

Различают локальное, удаленное представления и представление в виде веб-сервиса. В зависимости от вида представления различают локальный домашний интерфейс и локальный интерфейс или локальный бизнес-интерфейс для локального представления, удаленный домашний интерфейс и удаленный интерфейс или удаленный бизнес-интерфейс для удаленного представления. Эти классы принято называть EJB Local Home и EJB Local Object для локального представления, EJB Home и EJB Object для удаленного представления.

Локальное представление используется клиентами, которые исполняются в том же процессе виртуальной машины, что и сам компонент (локальный клиент). В роли локальных клиентов могут выступать EJB- или веб-компоненты этого же приложения.

Удаленное представление предназначено для клиентов, которые исполняются в другом процессе, нежели компоненты (удаленные клиенты). В роли удаленных клиентов могут вступать любые внешние Java- или CORBA-приложения, в том числе и те, которые используются на удаленных машинах (например, клиентские GUI-приложения, веб-, EJB-, CORBA-компоненты других приложений промежуточного слоя). Удаленное представление может быть использовано и локальными клиентами, при этом взаимодействие определенным образом оптимизируется и является более эффективным по сравнению с тем, что имеет место при использовании удаленных клиентов, однако оно значительно менее эффективно, чем при использовании локального представления. Поэтому для локальных клиентов рекомендуется всегда использовать локальное представление.

Между этими двумя представлениями имеется важное семантическое отличие, касающееся передачи параметров и возвращаемых значений вызываемыми методами компонентов.

При взаимодействии через локальное представление используются соглашения, принятые в языке Java, то есть значения простых типов передаются по значению (копированием в стек), а объекты – по ссылке. Это означает, что состояние параметров объектных типов может быть изменено внутри вызванного метода и это изменение будет доступно клиенту после возврата из метода.

При работе через удаленное представление все передается по значению. При этом передача объектов реализуется с помощью механизма сериализации, который является частью виртуальной машины и позволяет преобразовать объект в поток байтов, передать его в другую виртуальную машину и восстановить там объект из данного потока. При этом любые изменения объектов, сделанные внутри метода, не будут доступны клиенту после возврата из него.

Удаленное представление обеспечивает независимость клиентов от местоположения компонента, что позволяет гибко распределять компоненты по узлам вычислительной сети. Однако взаимодействие клиента с компонентом через удаленное представление характеризуется низкой эффективностью, так как велики накладные расходы на передачу запросов и ответов по сети и на сериализацию объектов. Поэтому методы удаленного представления, как правило, проектируют с ориентацией на уменьшение количества необходимых вызовов.

Разработчик должен обеспечить хотя бы одно представление для EJB-компонента. Определение локального и удаленного представлений для одного компонента возможно, но не рекомендуется, так как требуется уделять особое внимание отсутствию побочных эффектов при вызове через локальное представление.

Для компонентов EJB 3.0 можно обеспечить т.н. адаптированное представление, удовлетворяющее требованиям к представлению компонентов EJB 2.1 и позволяющее обеспечить доступ к компоненту из клиентов, отвечающих спецификации EJB 2.1. Адаптированное представление может быть удаленным или локальным и может дополнять EJB 3.0-представление или быть единственным представлением компонента.

Технология EJB, начиная с версии 2.1, обеспечивает средства интеграции с технологией веб-сервисов. Сессионный компонент без состояния может быть представлен как веб-сервис, необходимые для этого программные артефакты генерируются либо самим контейнером на этапе установки приложения либо специальным инструментарием, входящим в комплект поставки контейнера. В любом случае к компоненту, функционирующему как веб-сервис, необходимо разработать абстрактное описание в виде WSDL-документа (web service description language – это xml язык для абстрактного описания функциональности web-сервиса и некоторых специфических деталей в манере,

независящей от языка или программно-аппаратной платформы, на которой будет вестись реализация веб-сервиса и его компонентов). В WSDL-описание могут быть включены бизнес-методы компонента независимо от того, входят ли они в его компонентный интерфейс. По WSDL-описанию генерируется абстрактное описание сервиса на языке Java в виде интерфейса (т.н. Web Service Endpoint interface), а также классы-заглушки, классы, соответствующие xml-типам данных, и классы, реализующие сериализацию объектов этих типов в xml представление и наоборот.

Все сказанное выше о представлении справедливо для сессионных компонентов и компонентов сущностей. Взаимодействие с MDB выполняется асинхронно с помощью службы JMS (Java Message Service). Это означает, что бизнес-логика, реализуемая MDB, может быть активирована клиентом и исполняться параллельно с ним. Для активации логики и обмена данными используются сообщения, которые не ограничены рамками контейнера или машины. С точки зрения клиента MDB выглядит как JMS-очередь (queue) или тема подписки (topic). Фактически MDB – это средство интеграции технологии EJB в технологию JMS. Они позволяют вести разработку JMS-сервисов на основе компонентного подхода.