Веселовская А.Д., Лютаревич В.А.
Омский государственный университет имени Ф.М. Достоевского
Научный руководитель: *Никонова Г.В., доцент ОмГТУ*
Омский государственный технический университет (Омск)

ПРИОРИТЕТЫ В ИСПОЛЬЗОВАНИИ EJB КОНТЕЙНЕРА

Показаны достоинства EJB компонента в практическом применении: как его экземпляры управляют пулом, подключением к БД и прочее –, и почему действительно стоит его использовать.
Ключевые слова: EJB, Java, Java-контейнер, фреймворк.

EJB (Enterprise Java Beans) — это фреймворк, который используется для построения бизнес-логики приложения. Сервер приложений J2EE состоит из двух основных элементов:

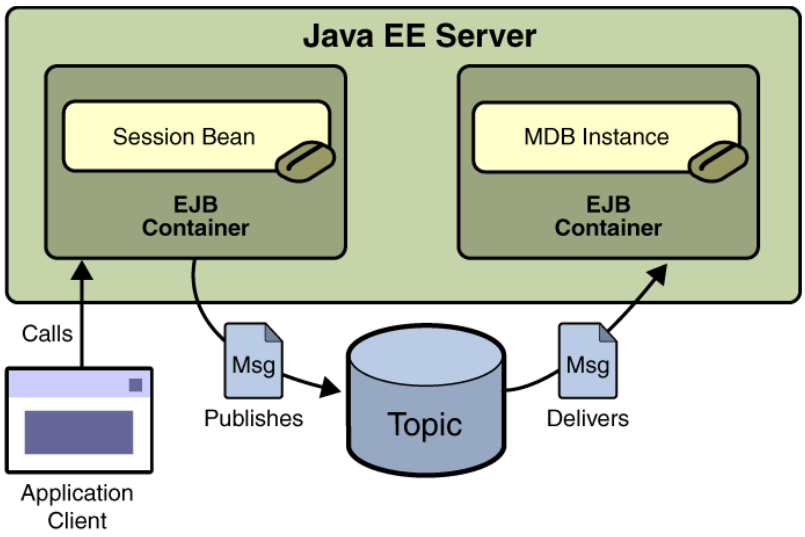
1. B-Container — (JSP, JSF и т.д.) все что дает конечный вид пользователю, а точней пользовательский интерфейс.
2. EJB-Container — используется для написания бизнес-логики.

С точки зрения EJB — это технология, предоставляющая множество готовых решений (управление транзакциями, безопасность, хранение информации и т.п.) для вашего приложения. Другими словами, каждый EJB-компонент является набором Java-классов со строго регламентированными правилами именования методов (верно для EJB 2.0, в EJB 3.0 за счет использования аннотаций выбор имён свободный)

EJB делится на три типа компонентов

1. Session beans — используется для построения бизнес-логики, которая может быть вызвана программным клиентом через локальный, удаленный или веб-интерфейс обслуживания клиентов.

Для доступа к приложению, развернутого на сервере, клиент вызывает методы сессионного компонента. Сессионный компонент выполняет работу для своего клиента, защищая его от сложности, выполняя бизнес-задач внутри сервера.



Существует 3 типа session-beans: singleton, stateless и stateful.

Stateful — автоматически сохраняют свое состояние между разными клиентскими вызовами.

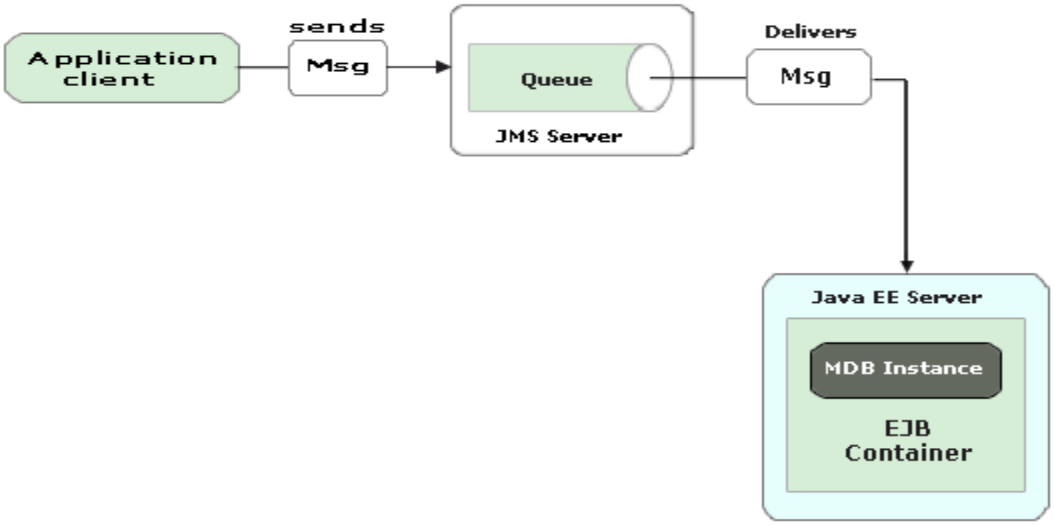
Stateless — без состояния, используются для реализации бизнесс-процессов, которые могут быть завершены за одну операцию.

Singleton — один объект на всё приложение (начиная с версии 3.1)

2. Message-Driven beans — компонент является корпоративным компонентом, который позволяет Java EE приложениям обрабатывать сообщения асинхронно.

Этот тип бинов обычно действует в качестве слушателя JMS-сообщения, который похож на слушателя событий, но получает JMS-сообщений вместо событий. Сообщения могут быть отправлены на любой компонент Java EE (клиентское приложение, другой компонент, или веб-компонент) или JMS приложение или систему, которая не использует Java EE технологий.

Message-Driven beans может обрабатывать не только JMS сообщения но и других видов сообщений.



На схеме выше можно наблюдать общение между приложением и сервером с помощью очереди куда поступают сообщения.

3. Entities — это сущности каких то объектов и в EJB оно является хранилищем данных на период жизненного цикла Entity.

Entities является свое-родным отображением таблиц в БД.

Одним из главных достоинств EJB 3.0 стал новый механизм работы с persistence context (*контекст постоянства*), он дает возможность автоматически сохранять объекты в реляционной БД используя технологию ORM.

Для работы с сущностями был создан JPA (Java Persistence API).

JPA определяет стандарт для:

1) конфигурации маппинга сущностей приложения и их отображения в таблицах БД;

2) EntityManager API — позволяет выполнять CRUD (create, read, update, delete) операции над сущностями;

3) Java Persistence Query Language (JPQL) — для поиска и получения данных приложения;

Почему действительно стоит использовать EJB фреймворк? Если вам нужен компонент, который обращается к базе данных, или получает доступ к другим ресурсам подключения/каталога,

или получает доступ из нескольких клиентов, или предназначен в качестве службы SOA (сервис-ориентированная архитектура), EJBs сегодня, как правило, "больше, сильнее, быстрее (или по крайней мере более масштабируемой) и проще ", чем POJOs (Plain Old Java Object). EJB является наиболее полезным и пригодным для обслуживания большого числа пользователей по веб-или корпоративной сети и несколько менее подходящим для небольших приложений внутри отделов компаний.

1. Многократное использование/совместное использование логики через многочисленные приложения/клиенты со слабой связью.

EJBs могут быть упакованы в собственных jar-файлах, развернуты и вызваны из множества мест. Они являются общими компонентами. POJOs тоже может быть (очень аккуратно) разработан как библиотека и упакован в jar-файлы, но EJBs поддерживают и локальный и удаленный доступ к сети - в том числе с помощью локального интерфейса Java, прозрачный RMI, JMS асинхронные сообщения и SOAP / REST веб-службы, отсутствие необходимости постоянно перемещать jar-файлы вручную с многократным развертыванием.

Они очень полезны для создания сервисов SOA. При использовании для локального доступа они являются POJOs (с добавлением свободных контейнерных служб). Дополнительное проектирования отдельного слоя EJB нужно для максимизации выполнения принципа инкапсуляцию, слабой связности компонентов, и реализации чистого интерфейса (Фасада), а также для отделения вызывающих сторон от сложных моделей обработки и передачи данных, тем самым соблюдается MVC паттерн.

2. Масштабируемость и надежность.

При получении сервером огромного количества запросов они будут распределены через доступные инстансы EJB в пуле и затем поставлены в очередь. Это означает, что если количество входящих запросов в секунду больше, чем сервер, может обработать, то мы эффективно боремся с переполнением – сервер всегда эффективно обрабатывает полученные запросы, а избыточные запросы устанавливаются в очередь, чтобы сервер мог обработать их, когда появится свободный экземпляр EJB в пуле. Таким образом сервер никогда попадет в «meltdown» - ситуация, при которой все полученные запросы испытывают очень большое время отклика одновременно, а сервер пытается получить доступ к большему количеству ресурсов, чем аппаратные средства и операционная система может предоставить и, следовательно, выходит из строя. EJBs может быть развернут на отдельном уровне, который может кластеризироваться - это дает системе отказоустойчивость (переход от одного сервера до другого), так же можно производить линейное масштабирование путем добавления дополнительных аппаратных средств.

3. Управление параллелизмом. Контейнер гарантирует, что EJB экземпляры автоматически обеспечивают потокобезопасный доступ несколькими клиентами одновременно. Он управляет EJB пулом, пулом потоков, очередью вызова, и автоматически блокирует метод записи (по умолчанию) и чтения (через @Lock (READ)) - это защищает данные от повреждений, возникающих при одновременной попытке двух и более экземпляров вызвать метод записи, а также делает возможным считывать данные последовательно, предотвращая затирание актуальных данных другими экземплярами во время чтения-записи.

В основном это полезно для @Singleton сеансовых компонентов, где компоненты манипулируют и совместно используют общее состояние через клиентские стороны. Всё это может быть легко переопределено, т.е. можно вручную настроить или программно контролировать дополнительные сценарии для параллельного выполнения кода и доступа к данным.

4. Автоматизированная обработка транзакций.

Все ваши методы EJB выполняются в транзакции JTA(Java Transaction API). При использовании подключения к базе данных посредством JPA или JBDC, оно автоматически добавляется в транзакцию. То же для JMS и вызовов JCA. Также необходимо указывать аннотацию @TransactionAttribute (someTransactionMode) перед методом, чтобы определить, принимает ли конкретный метод участие в

транзакции JTA, переопределяя режим по умолчанию: "Требуемый"

5. Очень простой доступ к ресурсу/зависимости через инъекции.

Контейнер будет искать ресурсы и устанавливать ссылки на ресурсы в качестве полей экземпляра в EJB: например, JNDI (Java Naming and Directory Interface) хранит JDBC подключения, JMS (Java Message Service) подключения/тема/очереди, другие EJBs, JTA транзакции, JPA (Java Persistence API) менеджер объекта персистентности контекста, JPA менеджер объекта фабрики персистентных юнитов, и JCA- стандарт архитектуры для соединения серверов приложений.

Для установки ссылок на другой EJB, JTA транзакции, JPA менеджер объектов и JMS, необходимо использовать код представленный ниже:

```
@Stateless
public class MyAccountsBean {

   @EJB SomeOtherBeanClass someOtherBean;
   @Resource UserTransaction jtaTx;
   @PersistenceContext(unitName="AccountsPU") EntityManager em;
   @Resource QueueConnectionFactory accountsJMSfactory;
   @Resource Queue accountPaymentDestinationQueue;

   public List<Account> processAccounts(DepartmentId id) {
      // Здесь можно использовать все вышеуказанные переменные без дополнительной установки.
      // Они автоматически принимают участие в транзакции JTA
   }
}
```

Сервлет может вызвать этот бит(компонент) локально, просто объявив переменную экземпляра:

```
@EJB MyAccountsBean accountsBean;
```

а затем вызывать нужные методы у данного экземпляра.

6. Умное взаимодействие с JPA.

По умолчанию, EntityManager, введенный как показано выше, использует persistence context в пределах транзакции. Это идеально для stateless session beans (*сессионных компонентов без сохранения состояния*). При вызове (stateless) EJB метода, в пределах новой транзакции создается новый persistence context, все экземпляры сущностей, полученные/записанные в БД, видны только в пределах этого вызова и изолированы от остальных методов. Если методом вызваны другие stateless EJB, контейнер распространяется на них и разделяет с ними тот же PC, так что все сущности автоматически становятся общими и согласованными через PC в той же транзакции.

Если session bean объявлен как @Stateful, та же слаженность достигается объявлением entityManager’a с расширенными границами:

@PersistentContent(unitName="AccountsPU, type=EXTENDED). Он существует все время жизни bean сессии, сквозь несколько вызовов бина и несколько транзакций, кэшируя в памяти ранее полученные/записанные копии сущностей ДБ, так что их не нужно доставать заново.

7. Контроль жизненного цикла.

Жизненный цикл EJB управляется контейнером. По требованию, он создает экземпляры EJB, очищает и инициализирует стадии stateful session bean’ов, пассивирует и активирует, вызывает callback-методы жизненного цикла, таким образом код EJB может участвовать в операциях жизненного цикла, получая и освобождая ресурсы, или выполнять другие действия, связанные с инициализацией и освобождением. Он также ловит все исключения, записывает их в лог, откатывает транзакции по требованию и бросает новые исключения EJB или @ApplicationExceptions.

8. Контроль безопасности.

Контроль доступа к EJB, основанный на ролях, может быть сконфигурирован через простую аннотацию или XML настройку. Сервер автоматически передает данные об аутентифицированном пользователе вместе с каждым вызовом в SecurityContext (principal и роль). Это гарантирует, что все RBAC правила автоматически исполняются, чтобы методы не могли быть вызваны не той ролью. Это позволяет EJB легко получать доступ к данным о пользователе/роли для дополнительных программных проверок. Это позволяет подключать дополнительную обработку безопасности (или даже IAM инструменты) к контейнеру обычным путем.

9. Стандартизация и переносимость.

EJB соответствует Java EE стандартам.и нормам, продвигая качество, легкость понимания и поддержку. Он также способствует переносимости кода на новые серверные решения, обеспечивая их поддержку тех же стандартов, и удерживает разработчиков от случайного перенимания проприетарных непереносимых решений.

10. Самое неожиданное: Простота.

Все описанное выше может быть исполнено с помощью хорошо упорядоченного кода - как используя настройки EJB в Java EE 6 по умолчанию, либо добавив пару аннотаций. Написание чего-либо промышленного уровня с помощью собственных “старых добрых Java-объектов” получится *гораздо* более громоздким, сложным и подверженным ошибкам процессом. Как только начинаешь писать с EJB, эти вещи становятся достаточно легкими в разработке, и вдобавок идет еще много всего полезного.

Оригинальная спецификация EJB 10-летней давности довольно сильно мешала продуктивности. EJB были раздутыми, требовали много кода и конфигураций, предоставляя всего около 2/3 преимуществ, написанных выше. Большинство веб-проектов даже их не использовали. Но ситуация существенно изменилась за 10 лет доводок, пересмотров, функционального усовершенствования и модернизации процесса разработки. В Java EE 6 они занимают высшую ступень в промышленных программных системах, предоставляя при этом простоту использования.

СПИСОК ЛИТЕРАТУРЫ

1. Oracle Technology Network [электронный ресурс] – Режим доступа: http://www.oracle.com/technetwork/java/javaee/ejb/index.html – Enterprise JavaBeans Technology (Дата обращения: 23.11.2016)

2. Семихатов, С. Краткое введение в технологию Enterprise JavaBeans [электронный ресурс] – Режим доступа: http://citforum.ru/internet/javascript/enterpjavabeans.shtml (Дата обращения: 21.11.2016)

3. Enterprise JavaBeans [электронный ресурс] – https://ru.wikipedia.org/wiki/Enterprise_JavaBeans (Дата обращения: 23.11.2016)