7. Компонентные системы

7.1 Основы компонентных программных систем

Компонентно-ориентированный подход к проектированию и реализации программных систем и комплексов является в некотором смысле развитием объектно-ориентированного подхода и практически более пригоден для разработки крупных и распределенных программных систем (например, корпоративных приложений).

С точки зрения компонентно-ориентированного подхода программная система – это набор компонентов с четко определенным интерфейсом. В отличие от других подходов программной инженерии, изменения в систему вносятся путем создания новых компонентов или изменения старых, а не путем рефакторинга существующего кода.

Программный компонент – это автономный элемент программного обеспечения, предназначенный для многократного использования, который может распространяться для использования в других программах в виде скомпилированного кода. Подключение к программным компонентам осуществляется с помощью открытых интерфейсов, а взаимодействие с программной средой осуществляется по событиям, причём в программе, использующей компонент, можно назначать обработчики событий, на которые умеет реагировать компонент.

Как видно из определения, применение компонентного программирования призвано обеспечить более простую быструю и прямолинейную процедуру первоначальной инсталляции прикладного программного обеспечения, а также увеличить процент повторного использования кода, т.е. усилить основные преимущества ООП.

Говоря о свойствах компонентов, следует, прежде всего, отметить, что это существенно более крупные единицы, чем объекты (в том смысле, что объект представляет собой конструкцию уровня языка программирования). Другими отличиями компонентов от традиционных объектов являются возможность содержать множественные классы и (в большинстве случаев) независимость от языка программирования.

Заметим, что, автор и пользователь компонента, вообще говоря, территориально распределены и, вполне возможно, они не только пишут программы, но и говорят на разных языках.

Можно выделить следующие основные преимущества применения компонентно-ориентированного подхода при проектировании и разработке ПО:

- снижение стоимости программного обеспечения;
- повышение повторного использования кода;
- унификация обработки объектов различной природы;
- менее человеко-зависимый процесс создания программного обеспечения.

Так как программный компонент подразумевает полноценное автономное использование в виде «черного ящика», к разработке программных компонентов предъявляются серьезные требования:

- *полная документированность интерфейса*: все методы, предоставляемые в интерфейсе программного компонента, должны быть качественно документированы, с учетом всех возможных вариантов их использования в сторонних приложениях;
- *тщательное тестирование:* необходимо учесть все возможные и невозможные варианты использования программного компонента в сторонних системах на всех возможных значениях входных данных;
- *тщательный анализ входных значений*: необходимо учитывать возможность передачи в программный компонент входных данных, не соответствующих его спецификации и адекватно обрабатывать такие ситуации;
- *возврат адекватных и понятных сообщений об ошибках:* так как один программный компонент может быть использован в большом числе сторонних программных систем, необходимо обеспечить сторонним разработчикам возможность получения информации об ошибках программного компонента и возможных вариантах их решения;
- *необходимость предусмотреть возможность неправильного использования.*

7.2 Концепция JavaBeans

JavaBeans — классы в языке Java, написанные по определённым правилам. Они используются для объединения нескольких объектов в один (bean) для удобной передачи данных. JavaBeans обеспечивают основу для многократно используемых, встраиваемых и модульных компонентов ПО.

Спецификация Sun Microsystems определяет JavaBeans, как «универсальные программные компоненты, которые могут управляться с помощью графического интерфейса».

Компоненты JavaBeans могут принимать различные формы, но наиболее широко они применяются в элементах графического пользовательского интерфейса. Одна из целей создания JavaBeans – взаимодействие с похожими компонентными структурами. Например, Windows-программа, при наличии соответствующего моста или объекта-обёртки, может использовать компонент JavaBeans так, будто бы он является компонентом COM или ActiveX.

7.2.1 Правила описания JavaBean

Чтобы класс мог работать как bean, он должен соответствовать определённым соглашениям об именах методов, конструкторе и поведении. Эти соглашения дают возможность создания инструментов, которые могут использовать, замещать и соединять JavaBeans.

Правила описания гласят:

- *Класс должен иметь public конструктор без параметров*. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
- *Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа)*, которые подчинятся стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
- *Класс должен быть сериализуем*. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.
- *Класс не должен содержать никаких методов обработки событий*.

Т.к. требования в основном изложены в виде соглашения, а не интерфейса, некоторые разработчики рассматривают JavaBeans, как Plain Old Java Object, которые следуют определённым правилам именования.

7.2.2 Enterprise JavaBeans

Enterprise JavaBeans – это высокоуровневая, базирующаяся на использовании компонентов технология создания распределенных приложений, которая использует низкоуровневый API для управления транзакциями. Первый вариант спецификации Enterprise JavaBeans появился в марте 1998 г. За время своего существования, технология прошла большой путь и продолжает развиваться.

Enterprise JavaBeans – больше, чем просто инфраструктура. Ее использование подразумевает еще и технологию (процесс) создания распределенного при-

ложения, навязывает определенную архитектуру приложения, а также определяет стандартные роли для участников разработки. Применение данных техник обеспечивает решение следующих стандартных проблем масштабируемых и эффективных серверов приложений с использованием Java:

- организация удаленных вызовов между объектами, работающими под управлением различных виртуальных машин Java;
- управление потоками на стороне сервера;
- управление циклом жизни серверных объектов (создание, взаимодействие с пользователем, уничтожение);
- оптимизация использования ресурсов (время процессора, память, сетевые ресурсы);
- создание схемы взаимодействия контейнеров и операционных сред;
- создание схемы взаимодействия контейнеров и клиентов, включая универсальные средства создания разработки компонентов и включения их в состав контейнеров;
- создание средств администрирования и обеспечение их взаимодействия с существующими системами;
- создание универсальной системы поиска клиентом необходимых серверных компонентов;
- обеспечение универсальной схемы управления транзакциями;
- обеспечение требуемых прав доступа к серверным компонентам;
- обеспечение универсального взаимодействия с СУБД.

Технология Enterprise JavaBeans определяет набор универсальных и предназначенных для многократного использования компонентов, которые называются Enterprise beans (далее – *компоненты EJB*). При создании распределенной системы, ее бизнес-логика будет реализована в этих компонентах. Каждый компонент EJB состоит из *удаленного интерфейса*, *собственного интерфейса* и *реализации EJB-компонента*. Удаленный интерфейс (remote-интерфейс) определяет бизнес-методы, которые может вызывать клиент EJB. Собственный (домашний) интерфейс (home-интерфейс) предоставляет методы `create` для создания новых экземпляров компонентов EJB, методы поиска (`finder`) для нахождения экземпляров компонентов EJB и методы `remove` для удаления экземпляров EJB. Реализация EJB-компонента определяет бизнес-методы, объявленные в удаленном интерфейсе, и методы создания, удаления и поиска собственного интерфейса.

После завершения разработки, наборы компонентов EJB помещаются в специальные файлы (архивы, jar), по одному или более компоненту, вместе со специальными *параметрами развертывания* (deployment). Затем они устанавливаются в специальной операционной среде, в которой запускается контейнер EJB. *Контейнер EJB* предоставляет окружение выполнения и средства управления жизненным циклом EJB-компонентам.

Клиент осуществляет поиск компонентов в контейнере с помощью home-интерфейса соответствующего компонента. После того, как компонент создан и/или найден, клиент выполняет обращение к его методам с помощью remote-интерфейса.

Контейнеры EJB выполняются под управлением *сервера* EJB, который является связующим звеном между контейнерами и используемой операционной средой. Сервер EJB обеспечивает доступ контейнерам EJB к системным сервисам, таким как управление доступом к базам данных или мониторинг транзакций.

Все экземпляры компонентов EJB выполняются под управлением контейнера EJB. Контейнер предоставляет системные сервисы размещенным в нем компонентам и управляет их (компонентов) жизненным циклом. В общем случае контейнер предназначен для решения следующих задач:

– *Обеспечение безопасности*. Дескриптор развертывания (deployment descriptor) определяет права доступа клиентов к бизнес-методам компонентов. Обеспечение защиты данных обеспечивается за счет предоставления доступа только для авторизованных клиентов и только к разрешенным методам.
– *Обеспечение удаленных вызовов*. Контейнер берет на себя все низкоуровневые вопросы обеспечения взаимодействия и организации удаленных вызовов, полностью скрывая все детали процесса, как от разработчика компонентов, так и от клиентов. Это позволяет производить разработку компонентов точно так же, как если бы система работала в локальной конфигурации, т.е. вообще без использования удаленных вызовов.
– *Управление жизненным циклом*. Клиент создает и уничтожает экземпляры компонентов, однако контейнер для оптимизации ресурсов и повышения производительности системы может самостоятельно выполнять различные действия, например, активизацию и деактивацию этих компонентов, создание их пулов и т.д.
– *Управление транзакциями*. Все параметры, необходимые для управления транзакциями, помещаются в дескриптор поставки. Все вопросы по обес-

печению управления распределенными транзакциями в гетерогенных средах и взаимодействия с несколькими базами данных берет на себя контейнер EJB. Контейнер обеспечивает защиту данных и гарантирует успешное подтверждение внесенных изменений; в противном случае транзакция откатывается.

7.2.3 Типы компонентов EJB

Существуют два типа компонентов EJB: сессионные и сущностные компоненты (session- и entity-компоненты).

Сессионный компонент представляет собой объект, созданный для обслуживания запросов *одного* клиента. В ответ на удаленный запрос клиента контейнер создает экземпляр такого компонента. Сессионный компонент всегда сопоставлен с одним клиентом, и его можно рассматривать как «представителя» клиента на стороне EJB-сервера. Такие компоненты могут «знать» о наличии транзакций – они могут отвечать за изменение информации в базах данных, но сами они непосредственно не связаны с представлением данных в БД.

Сессионные компоненты являются временными объектами. Обычно сессионный компонент существует, пока создавший его клиент поддерживает с ним сеанс связи. После завершения связи с клиентом компонент уже никак не сопоставляется с ним. Объект считается временным, так как в случае завершения работы или краха сервера клиент должен будет создать новый компонент.

Обычно сессионный компонент содержит параметры, которые характеризуют состояние его взаимодействия с клиентом, т.е. он сохраняет некоторое состояние между вызовами удаленных методов в процессе сеанса связи с клиентом.

Сущностные компоненты представляют собой объектное представление данных из базы данных. Например, компонент сущности может моделировать одну запись из таблицы реляционной базы данных (или набор записей из связанных таблиц). Ключевым отличием сущностного компонента от сессионного является то, что несколько клиентов могут одновременно обращаться к одному экземпляру сущностного компонента. Сущностные компоненты изменяют состояние сопоставленных с ними баз данных в контексте транзакций.

Состояние компонентов-сущностей в общем случае нужно сохранять, и живут они столько, сколько существуют в базе данных те данные, которые они представляют, а не столько, сколько существует клиентский или серверный процесс. Остановка или крах контейнера EJB не приводит к уничтожению содержащихся в нем сущностных компонентов.

7.2.4 Составные части EJB-компонента

EJB-компонент физически состоит из нескольких частей, включая сам компонент, реализацию некоторых интерфейсов и информационный файл. Все это собирается вместе в специальный jar-файл – модуль развертывания.

- *Enterprise Bean* является Java-классом, разработанным поставщиком Enterprise Bean. Он реализует интерфейс Enterprise Bean и обеспечивает реализацию бизнес-методов, которые выполняет компонент. Класс не реализует никаких методов авторизации, многопоточности или поддержки транзакций.
- *Домашний интерфейс*. Каждый создающийся Enterprise Bean должен иметь ассоциированный домашний интерфейс. Домашний интерфейс применяется как фабрика для компонента EJB. Клиент использует домашний интерфейс для нахождения экземпляра компонента EJB или создания нового экземпляра компонента EJB.
- *Удаленный интерфейс* является Java-интерфейсом, который отображает через рефлексию те методы Enterprise Bean, которые необходимо показывать внешнему миру. Удаленный интерфейс играет ту же роль, что и IDL-интерфейс в CORBA, и обеспечивает возможность обращения клиента к компоненту.
- *Описатель развертывания* является XML-файлом, который содержит информацию относительно компонента EJB. Использование XML позволяет установщику легко менять атрибуты компонента. Конфигурационные атрибуты, определенные в описателе развертывания, включают:
  - имена домашнего и удаленного интерфейса;
  - имя JNDI для публикации домашнего интерфейса компонента;
  - транзакционные атрибуты для каждого метода компонента;
  - контрольный список доступа для авторизации.
- *EJB-Jar-файл* – это обычный java-jar-файл, который содержит компонент (компоненты) EJB, домашний и удаленный интерфейсы, а также описатель развертывания.

7.2.5 Инфраструктура Enterprise JavaBean

Инфраструктура EJB обеспечивает удаленное взаимодействие объектов, управление транзакциями и безопасность приложения. Спецификация EJB оговаривает требования к элементам инфраструктуры и определяет Java Application Programming Interface (API); она не касается вопросов выбора платформ, протоколов и других аспектов, связанных с реализацией.

Ниже показаны различные элементы инфраструктуры EJB. Она должна обеспечивать канал связи с клиентом и другими компонентами EJB. Инфраструктура должна также обеспечить соблюдение прав доступа к компонентам EJB.

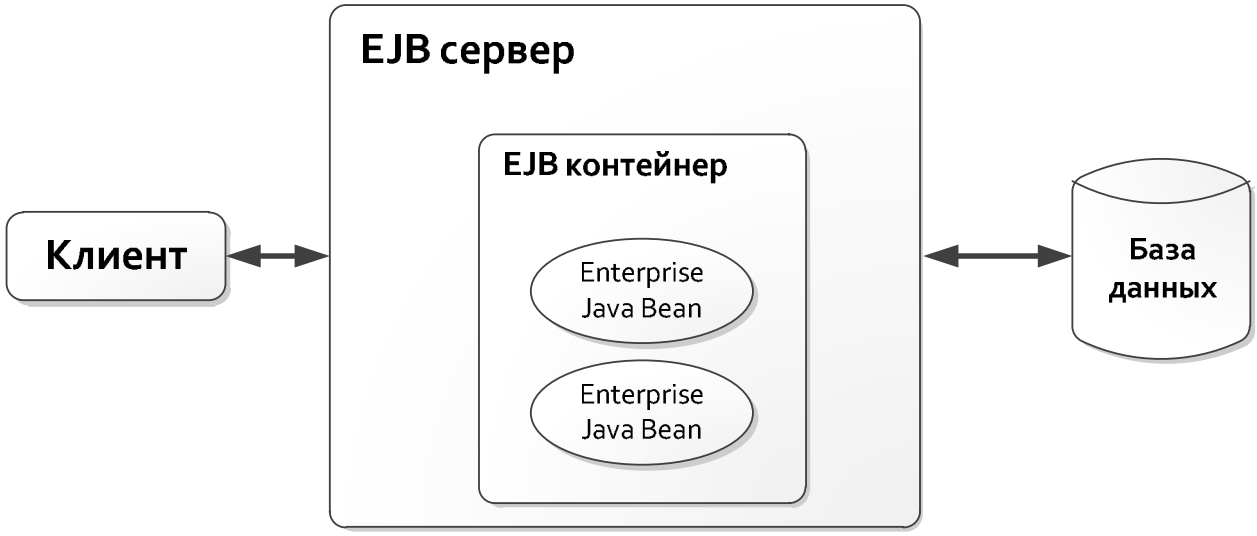

Рис. 22. Компоненты, контейнеры и серверы EJB

В общем случае необходимо гарантировать сохранение состояния компонентов в контейнерах. Инфраструктура EJB обязана предоставить возможности для интеграции приложения с существующими системами и приложениями – без этого нельзя говорить о пригодности приложения для функционирования в сложной информационной среде. Все аспекты взаимодействия клиентов с серверными компонентами должны происходить в контексте транзакций, управление которыми возлагается на инфраструктуру EJB. Для успешного выполнения процесса поставки компонентов инфраструктура EJB должна обеспечить возможность взаимодействия со средствами управления приложениями.

Таким образом, спецификация Enterprise JavaBeans – это существенный шаг к стандартизации модели распределенных объектов в Java. В настоящее время существует большое количество инструментов, поддерживающих этот подход и помогающих ускорить разработку EJB-компонентов.