Введение в платформу Java EE

Эволюция систем обработки данных

Java Platform, Enterprise Edition (Java EE) – это платформа для разработки и выполнения приложений промежуточного уровня (middleware), которые располагаются между ресурсами (БД, почтовыми и файловыми системами, системами асинхронной обработки сообщений и т.д.) и клиентскими приложениями, и обеспечивают последним прикладной, то есть ориентированный на решение определенного класса задач, программный интерфейс доступа к ресурсам и средствам обработки данных.

Необходимость в промежуточном уровне обусловлена объективными причинами, а трехуровневые системы (ресурсы – middleware – клиент) есть результат эволюции архитектуры корпоративных систем обработки данных.

Для лучшего понимания особенностей различных архитектур информационных систем разделим приложения на типовые функциональные компоненты[1]:

- PS (Presentation Services) - средства представления. Обеспечиваются устройствами, принимающими ввод от пользователя и отображающим то, что сообщает ему компонент логики представления PL, плюс соответствующая программная поддержка. Может быть текстовым терминалом или X-терминалом, а также ПК или рабочей станцией в режиме программной эмуляции терминала или X-терминала;
- PL (Presentation Logic) - логика представления. Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя по выбору альтернативы меню, по нажатию кнопки или при выборе элемента из списка;
- BL (Business or Application Logic) - прикладная логика. Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение;
- DL (Data Logic) - логика управления данными. Операции с базой данных (SQL-операторы SELECT, UPDATE и INSERT), которые нужно выполнить для реализации прикладной логики управления данными;
- DS (Data Services) - операции с базой данных. Действия СУБД, вызываемые для выполнения логики управления данными, такие как манипулирование данными, определения данных, фиксация или откат транзакций и т. п.;
- FS (File Services) - файловые операции. Дисковые операции чтения и записи данных для СУБД и других компонент. Обычно являются функциями ОС.

Рассмотрим типовые архитектуры корпоративных систем обработки данных в порядке их эволюции.

1. *Централизованные системы*.

Данные, приложения и средства взаимодействия с пользователем сосредоточены в большой ЭВМ, которая выполняет абсолютно все операции по хранению и обеспечению доступа к данным (DL, DS, FS), их прикладной обработке (BL) и взаимодействию с пользователем (PL). Терминалы обеспечивают лишь возможность ввода/вывода (PS): набранные пользователем символы по последовательной линии поступают в ЭВМ, а ответ (тоже символьный) передается терминалу для отображения на экране. Терминалы не обладают вычислительными способностями.

Достоинство: простота обслуживания и сопровождения ПО и средств ВТ.

Недостатки:

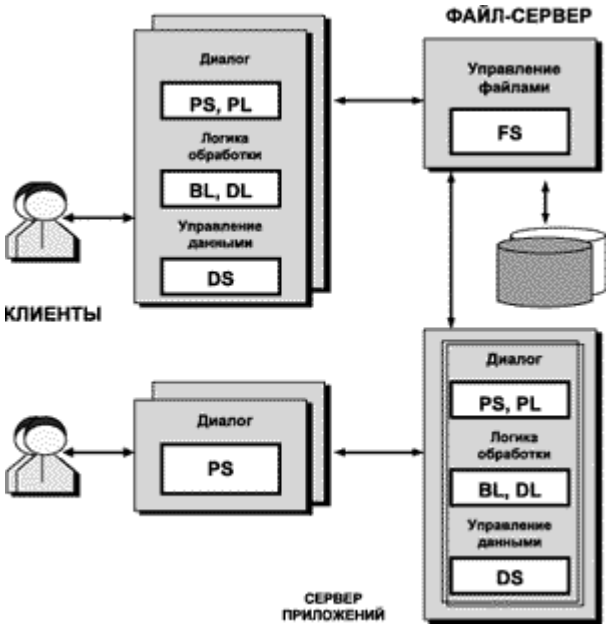
1) высокая стоимость решения, поскольку машина должна быть достаточно мощной, чтобы обеспечить комфортную работу возможно нескольких десятков пользователей и приложений;

---

[1] См. Артемьев В.И. Обзор способов и средств построения информационных приложений // «Системы управления базами данных», 1996, №5. – Постоянный адрес статьи: http://www.osp.ru/dbms/1996/05/52.htm

2) каждый дополнительный пользователь и приложения вносят существенную нагрузку, что приводит к потере масштабируемости;
3) трудность обеспечения графического интерфейса;
4) затруднена организация распределенной обработки информации.

2. *Файл-серверные системы*.



Файловый сервер хранит прикладные данные в виде системы файлов и позволяет обращаться к этим файлам по сети (FS), обработка информации (PS, PL, BL, DL, DS) ведется на рабочих станциях средствами клиентских прикладных программ, например, для расчета заработной платы, управления запасами на складе и т.п.

Данные хранятся в табличных файлах (например, типа DBF), которые могут быть проиндексированы (файлы типа IDX, MDX, CDX). Объектами разработки в файл-серверном приложении являются компоненты приложения, определяющие логику диалога PL, а также логику обработки BL и управления данными DL. Разработанное приложение реализуется либо в виде законченного загрузочного модуля или в виде специального кода для интерпретации средствами «настольных» СУБД типа DBase, FoxPro, Clipper, Paradox.

Достоинства:

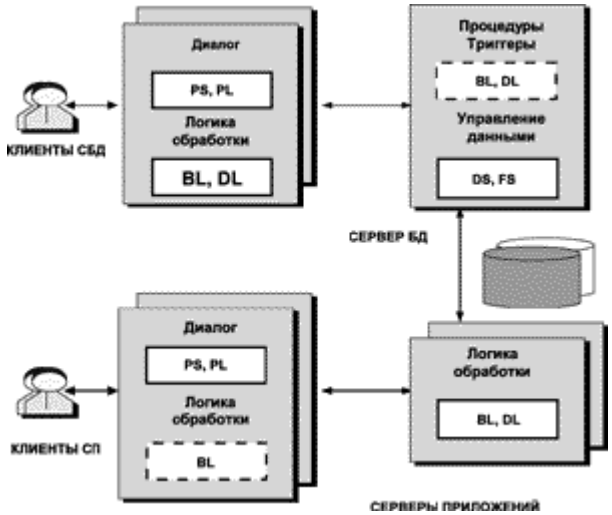
1) появилась возможность распределенной обработки информации, так как обработка идет на каждой рабочей станции, как следствие, каждый новый клиент добавляет вычислительную мощность к сети;
2) более дешевое решение, поскольку используется недорогие аппаратные средства.

Недостатки:

1) так как работа ведется с файлом средствами сетевых клиентов (на рабочих станциях) и файловых служб ОС (на сервере), то это приводит к постоянному обмену фрагментами файлов, которые передаются от сервера к клиентам. Поскольку вся обработка данных ведется на клиенте, то большая часть полученных им данных может не использоваться (например, при любых операциях поиска и фильтрации). Возникает большая нагрузка на сеть, которая быстро становится узким местом при росте количества клиентов. Этот эффект усугубляется и используемыми на тот момент сетевыми технологиями: топология типа «шина» на базе Ethernet 10Base5, 10Base2 с пропускной способностью до 10 Мбит/с;
2) так как работа ведется с файлами, а не с данными, на файловом сервере нет возможности организовать централизованное управление множественным доступом к данным и автоматическое разрешение возможных при этом коллизий. Фактически блокирование нужных элементов БД одним клиентом и проверка блокирования этих же элементов другим выполняется на уровне приложений, выполняющихся на рабочих

станциях, а не сервером. Такая архитектура сильно затрудняет разработку пользовательских приложений и в общем случае не позволяет решить проблему целостности и согласованности данных при множественном доступе.

3. *Клиент-серверные системы*.



Архитектура клиент-сервер предназначена для разрешения проблем файл-серверных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать более эффективно.

Данное решение предоставляет доступ к данным не как к файлам, а как к некоторому набору абстрактных структур (деревья, таблицы), при этом физический способ представления данных в том виде, в котором они хранятся на сервере, клиенту не доступен и не важен. Доступ к данным предоставляется не файловой службой ОС, а специальной службой сервера БД. Эта служба принимает от клиентов по сети запросы к БД, выраженные на некотором специальном языке в терминах соответствующей модели данных. Получив запрос, сервер БД обрабатывает его и возвращает клиенту результат (данные выборки, количество обновленных записей, код ошибки).

Логику обработки данных приходится совмещать с интерфейсом пользователя или с БД. Последнее выполняется с помощью хранимых процедур и триггеров, которые разрабатываются на том или ином процедурном расширении языка SQL, сохраняются и исполняются в составе сервера БД.

Достоинства:

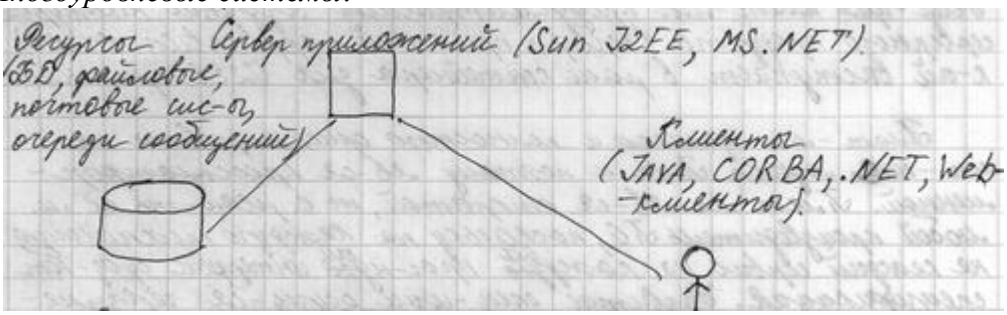
1) балансировка нагрузки на вычислительные узлы и сеть. За счет того, что основная работа по базовой обработке данных возложена на сервер БД, рабочие станции могут быть маломощными и дешевыми по сравнению с сервером. Кроме того, поскольку сервер БД возвращает лишь результат обработки запроса, а сама обработка выполняется им локально, то это позволяет минимизировать количество обращений клиентов к серверу и объем сетевого трафика;
2) так как все запросы выражены в терминах данных и обрабатываются централизованно, то появляется возможность эффективно управлять множественным доступом к данным и надежно разрешать возникающие при этом коллизии. Причем все это реализуется не клиентскими приложениями, а сервером БД на основе механизма транзакций, что упрощает разработку клиентских приложений;
3) данные отделены от приложений, работающих с ними, и могут поддерживаться относительно независимо друг от друга.

Недостатки:

1) сложность сопровождения вследствие необходимости администрирования приложений для большого числа клиентов при отсутствии унификации в конфигурациях клиентов и средств управления изменениями;
2) сложность сопровождения как следствие сложности отладки хранимых процедур и триггеров, а также их непереносимости между СУБД различных поставщиков;

3) чрезмерное использование хранимых процедур для реализации прикладной логики снижает масштабируемость сервера;
4) возрастает время реакции из-за ожидания завершения пакетного задания на сервере и влияния таких заданий на диалоговых пользователей;
5) проблема обеспечения целостности распределенной транзакции в неоднородной распределенной БД.

4. *Многоуровневые системы*.



Ресурсы предоставляют данные (DS, FS). Сервер приложений содержит прикладную логику обработки данных (BL, DL). Клиенты позволяют отображать данные и реагировать на действия пользователя (PS, PL).

Достоинства:

1) независимость клиентских приложений от места расположения, физической природы и логической структуры ресурсов;
2) гибкие возможности по сопровождению слоев приложения и его администрированию: модернизацию ресурсов и реализацию компонентов middleware можно производить, не затрагивая клиентов, равно как и клиентов можно обновлять независимо от бизнес-логики;
3) обеспечение целостности распределенной транзакции в неоднородной распределенной системе;
4) у приложений может быть множество различных представлений (веб-интерфейс, интерфейс командной строки, графический интерфейс пользователя) при сохранении единой бизнес-логики.

Недостатки:

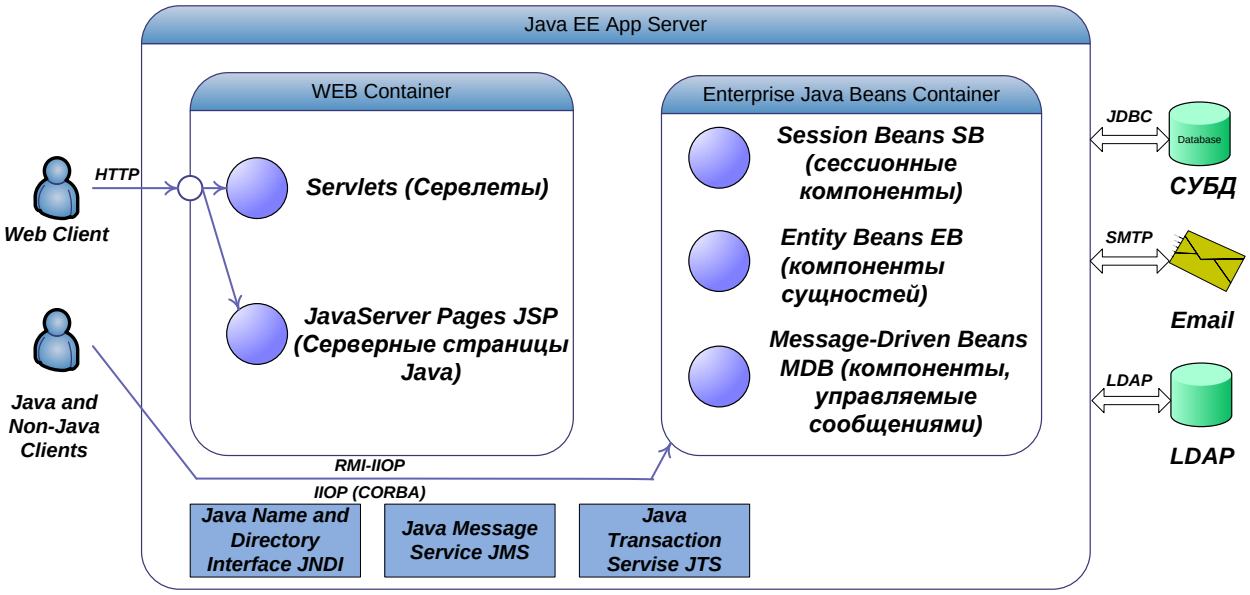
1) поскольку границы между компонентами PL, BL и DL размыты, прикладная логика может появиться на всех трех звеньях;
2) внедрению таких решений препятствует их относительная дороговизна.

Общая характеристика платформы Java EE

Java EE является одной из платформ, предназначенных для разработки и развертывания middleware приложений. Эта платформа определяет компонентную модель приложений и набор сервисов, которые платформа предоставляет компонентам. Реализация платформы предусматривает создание некоторого серверного продукта (сервера приложений – application server), который выступает в роли контейнера для Java EE-приложений.

Платформа целиком и полностью ориентирована на использования языка Java и поэтому является кроссплатформенной. Платформа является открытой, то есть реализовать ее может любой производитель ПО, поскольку на каждую технологию, сервис, программный интерфейс существует спецификация. Развитие спецификации осуществляется через механизм Java Community Process (JCP), который позволяет привлечь к этому процессу всех заинтересованных производителей ПО.

В настоящее время платформа Java EE является стандартом де факто для индустрии больших программных систем. Обобщенно сервер приложений Java EE можно изобразить так:



Java EE-приложения состоят из компонентов различных видов, при этом компоненты могут быть распределены по разным машинам, а обращение к ним может выполняться удаленно. Платформа обеспечивает весь необходимый комплекс средств для построения распределенных приложений. Java EE-компоненты выполняются в рамках некоторой среды, которая обеспечивается контейнером. Контейнер предоставляет содержащимся в нем компонентам различные сервисы:

1) управление жизненным циклом (lifecycle management);
2) координация распределенных транзакций (distributed transaction coordination/processing DTP);
3) сервис имен (naming service);
4) сервис обмена сообщениями (messaging service) и др.

Важнейшим сервисом является сервис управления ЖЦ. Он состоит в том, что любой компонент создается и уничтожается контейнером, а не пользовательским приложением. Во время существования компонент может пребывать в различных состояниях. На различных этапах ЖЦ контейнер вызывает те или иные методы экземпляров компонентов. Эти методы компоненты обязаны реализовывать. Контейнер поддерживает связанную с компонентом объектную инфраструктуру, которую компонент может использовать для реализации своих функций.

Сервис координации распределенных транзакций необходим для того, чтобы обеспечить синхронизацию начала и окончания глобальной транзакции, охватывающей возможно несколько разнородных источников данных.

Сервис имен позволяет по имени получить доступ к ресурсу того или иного типа. Он используется, например, для поиска компонентов, источников данных, очередей сообщений и пр.

Сервис обмена сообщениями используется для организации асинхронного взаимодействия между компонентами и приложениями.

Промежуточное ПО решает разнообразные задачи, поэтому платформа предусматривает компоненты различных видов, которые отличаются ЖЦ, характером взаимодействия с клиентами и набором доступных сервисов.

Выделяют две большие группы компонентов, поддерживающие различные варианты построения приложений на их основе и клиентов:

1) Enterprise Java Beans (EJB) – невизуальные компоненты, предназначенные для реализации бизнес-логики приложений;
2) веб-компоненты (web components) – компоненты, предназначенные для реализации веб-интерфейса приложений.

Так как существует два вида компонентов, то существует и два типа контейнеров: EJB-контейнер обеспечивает среду выполнения для EJB-компонентов, веб-контейнер – для веб-компонентов, при этом возможны различные реализации платформы. Например, можно реализовать веб-контейнер в виде отдельного продукта (пример – Apache Tomcat), можно реализовать отдельно EJB-контейнер (пример – JBoss), а можно реализовать и то, и другое в составе одного Java EE-сервера.

Краткая характеристика компонентов

Веб-компоненты бывают двух видов:

1) сервлеты (servlets);

2) серверные страницы Java (JavaServer Pages – JSP).

И тот, и другой вид предназначены для реализации функциональности, которая может быть активирована по веб-протоколам (HTTP, HTTP over SSL). Работа с этими компонентами ведется через веб-браузер (если функциональность ориентирована на интерактивное использование конечным пользователем). Возможно также использование этой функциональности как программного интерфейса, ориентированного на передачу данных в соответствии с архитектурой веб-сервисов.

Сервлеты могут использоваться в двух качествах:

1) front end component;

2) presentation component.

В первом случае сервлет выступает в роли так называемого контроллера (servlet controller) и координирует работу различных компонентов приложений, так как все веб-запросы к приложению проходят через него. Сервлет обеспечивает извлечение параметров запроса, вызов методов бизнес-логики, реализованных в виде EJB-компонентов или Java-классов, и перенаправление запроса на нужную JSP-страницу для отображения полученных от бизнес-методов данных.

Такой подход позволяет исключить из сервлета и бизнес-логику, и шаблон HTML-страницы. Бизнес-логика реализуется отдельно (например, в виде EJB-компонентов), а вывод информации реализуется средствами JSP-страниц.

Если сервлет используется во втором качестве, то задача сервлета – сгенерировать окончательный ответ (например, HTML-страницу). Данный подход не рекомендуется к использованию Соответствующие задачи решаются с помощью JSP. Исключение составляют случаи, когда генерируемый ответ не текстовый, а бинарный (например, изображение или аудиофайл).

Технология JSP является надстройкой над технологией сервлетов, она ориентирована на создание визуальной части веб-интерфейса путем декларативного описания генерируемых веб-страниц. В отличие от сервлетов JSP-страницы рекомендуется использовать в качестве presentation components. Хотя встречаются решения, когда JSP используется как контроллер.

В составе технологии JSP есть набор средств для описания шаблона генерируемой страницы и внедрения туда действий (actions), отрабатываемых на этапе обработки запроса. JSP-страница в результате состоит из элементов, часть которых (шаблон, template) является статической и передается клиенту без трансформации, а часть отрабатывается на сервере, генерируя динамическую часть ответа.

EJB-компоненты бывают трех видов:

1) сессионные компоненты (session beans);
2) компоненты сущностей (entity beans);
3) компоненты, управляемые сообщениями (message-driven beans, или MD-компоненты).

Сессионные компоненты делятся на два подвида:

1) сессионные компоненты без состояния (stateless session beans – SSB);

2) сессионные компоненты с состоянием (stateful session beans – SFSB).

SSB-компоненты предназначены для реализации бизнес-операций, между обращениями к которым не нужно запоминать никакой информации (возможно специфичной для того или иного клиента). Эти компоненты моделируют действия и не обладают состоянием. Если Java EE-приложения содержат EJB-слой, то SSB-компоненты присутствуют обязательно.

SFSB-компоненты моделируют процесс и обладают состоянием, которое специфично для каждого конкретного клиента. Это состояние актуально, пока длится сеанс работы клиента с компонентом. Этот сеанс может быть прерван сервером приложений в одностороннем порядке по истечении некоторого (настраиваемого) тайм-аута, в течение которого клиент не обращается к компоненту.

Entity-компоненты обладают состоянием, которое автоматически синхронизируется с неким постоянным хранилищем (чаще всего это реляционная БД). При этом клиент работает с экземпляром компонента как с объектом, вызывает его методы, которые изменяют его состояние. За синхронизацию состояния с источником данных отвечает контейнер. Entity-компоненты позволяют работать с данными хранилища на компонентном уровне.

MD-компоненты позволяют организовать асинхронное взаимодействие между компонентами одного или нескольких приложений. Функционирование MD-компонентов поддерживается сервисом обмена сообщениями (Java Message Service – JMS), средствами которого обеспечивается асинхронная передача данных. MD-компоненты позволяют организовать обработку асинхронных сообщений, используя компонентный подход.

Особенности разработки Java EE-приложений

Любое Java EE-приложение представляет собой некоторую систему компонентов, которая должна быть собрана определенным образом для того, чтобы ее как единое приложение можно было установить на сервер приложений. При этом жизненный цикл Java EE-приложения предусматривает соответствующий этап установки (deployment). Для поддержки этого этапа любая реализация платформы предусматривает специальные средства, которые среди прочего позволяют выполнять его удаленно.

Java EE-приложение перед его установкой физически представляется в виде архива с расширением `.ear` (enterprise archive), который содержит один или несколько модулей установки (deployment unit). Модули установки могут быть разных видов. Далее будем рассматривать два из них:

1) веб-модули (web-unit, web-war);

2) EJB-модули (ejb-unit, ejb-jar).

Любой модуль физически представляет собой архив, внутри которого должна существовать определенная структура каталогов. В модуль помещаются классы компонентов в скомпилированной форме, а также вспомогательные классы и файлы. Кроме того, любой модуль должен содержать один или несколько XML-дескрипторов.

Веб-модуль физически представляет собой архив с расширением `.war` (web archive) и содержит веб-компоненты и веб-контент (html-страницы, графика gif/jpeg/png, стили CSS, модули JavaScript и т.д.). EJB-модуль – это архив с расширением `.jar` (java achive), который содержит EJB-компоненты и связанные с ними классы.

Современные серверы приложений допускают установку не только приложений в виде `.ear`-архивов, но и непосредственно независимых (standalone) веб- и EJB-модулей.

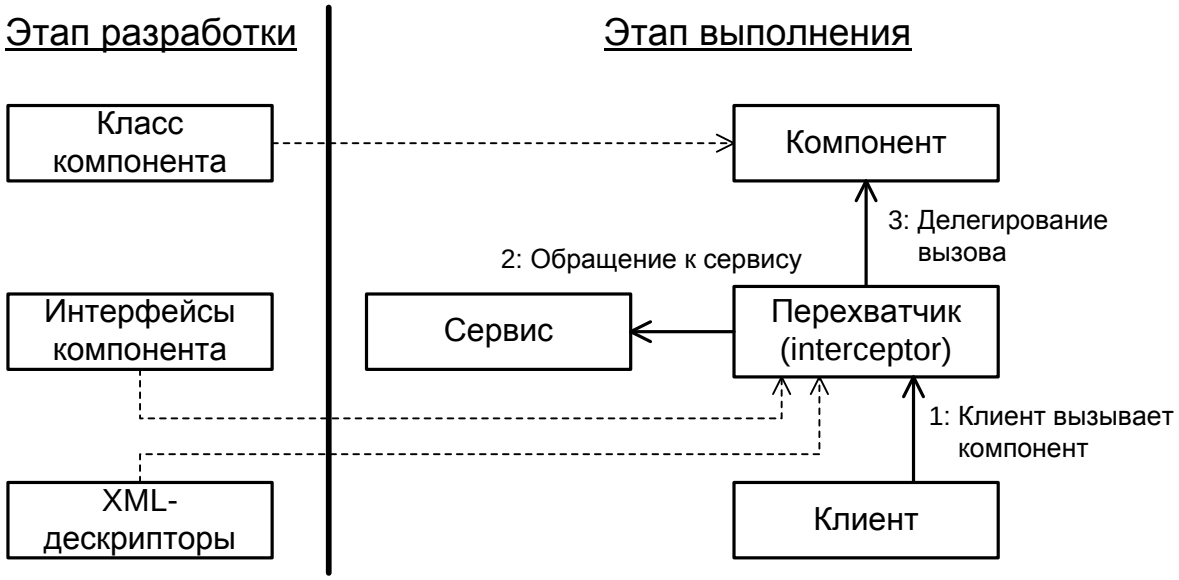
К каждому Java EE-приложению и каждому модулю прикладывается свой XML-дескриптор, который описывает в стандартном виде содержимое приложения или модуля. Кроме того, дескрипторы регламентируют целый круг вопросов, связанных с развертыванием компонентов приложения на сервере приложений. В частности, некоторые аспекты их поведения, аспекты, связанные с использованием сервисов, аспекты

безопасности и множество других. Спецификациями закреплены структура и содержание стандартного дескриптора Java EE-приложения, веб-модуля и EJB-модуля.

Спецификации платформы оставляют ряд вопросов на откуп разработчикам реализаций, которые регламентируются специфическими (custom) дескрипторами, структура и содержание которых зависят от конкретного сервера приложений. К этому кругу вопросов относятся описание соответствия между структурой entity-компонентов и структурой источника данных, с которыми их состояние будет синхронизироваться.

Дескрипторы важны для платформы Java EE, поскольку на их основе реализуется концепция неявного промежуточного ПО (implicit middleware). Суть этой концепции состоит в том, что разработчик компонента концентрируется на бизнес-логике, специфичной для его приложения. Код рутинных операций, связанных с обращением к сервисам платформы программистом при этом не пишется, он генерируется сервером приложений на этапе установки приложения. Дескрипторы при этом играют ведущую роль, определяя многие важнейшие аспекты этого генерируемого кода.

Достоинство такого подхода в том, что гораздо проще описать рутинные операции декларативно в дескрипторах, чем программировать их самостоятельно каждый раз. Таким образом, дескриптор содержит информацию о сервисах, необходимых компоненту и о порядке взаимодействия с ним, и это влияет на код, генерируемый на этапе установки приложения. Обобщенно это можно представить так:



В Java SE 5 появилась возможность указывать метаданные для объявляемого пакета, класса, метода, поля или локальной переменной с помощью аннотаций. Аннотации могут быть доступны не только на этапе компиляции, но и на этапе выполнения Java-приложения через рефлексию. Это позволяет использовать комбинацию XML-дескрипторов и аннотаций для развертывания Java EE-приложения.