Введение в Java EE

Сегодня поговорим о том, что это такое — Java EE: из чего состоит, каковы особенности архитектуры Java EE приложений, и приведем описания различных технологий данной платформы. Тема обширная сама по себе, но на азах мы не остановимся. В конце проведем небольшое сравнение Java EE со Spring Framework и ответим на вопрос: “что лучше учить” (спойлер: конечно же, учить нужно всё =))

Начнем с основ.

Java EE — что это?

Java EE — это платформа, построенная на основе Java SE, которая предоставляет API и среду времени выполнения
для разработки и запуска крупномасштабных, многоуровневых, масштабируемых, надежных и безопасных сетевых приложений.

Подобные приложения называют корпоративными (Enterprise applications), так как они решают проблемы, с которыми сталкиваются большие бизнесы.

Однако использовать подобные приложения и преимущества, которые дает Java EE, могут не только крупные корпорации и правительственные структуры. Решения, которые предлагает платформа Java EE, полезны, а порой просто необходимы отдельным разработчикам и небольшим организациям.

Развитие Java EE

Java EE развивается по процессу Java Community Process (JCP), сформированному в 1998 году. Он позволяет заинтересованным лицам участвовать в формировании будущих версий спецификаций платформ языка Java.

Основу данного процесса составляют JSR (Java Specification Request — запрос на спецификацию Java), формальные документы, описывающие спецификации и технологии, которые предлагается добавить к Java-платформе. Подобные запросы составляются членами сообщества — простыми разработчиками и

компаниями. К числу последних относятся Oracle, Red Hat, IBM, Apache и многие другие.

Т.е. ребята предлагают на рассмотрение новые фичи и плюшки, которые они хотели бы включить в Java. А затем проводят голосование, на основании которого принимается решение, что включить в следующую версию.

История версий Java EE выглядит так:

- J2EE 1.2 (Декабрь 1999)
- J2EE 1.3 (Сентябрь 2001)
- J2EE 1.4 (Ноябрь 2003)
- Java EE 5 (Май 2006)
- Java EE 6 (Декабрь 2009)
- Java EE 7 (Май)
- Java EE 8 (Август 2017)
- Jakarta EE 8 (Сентябрь 2019)

В 2017 году произошла новая веха в развитии платформы: Oracle передал контроль над развитием Java EE организации Eclipse Foundation. А в апреле 2018 года Java EE переименовали в Jakarta EE, которая полностью совместима с Java EE 8.

Архитектура Java EE приложений

Небольшое введение. Чтобы облегчить восприятие, давайте поговорим об устройстве Java EE приложений и некоторых терминах, которые мы будем употреблять далее. У Java EE приложений есть структура, которая обладает двумя ключевыми качествами:

- во-первых, многоуровневость. Java EE приложения — многоуровневые, и об этом мы еще поговорим подробнее;
- во-вторых, вложенность. Есть Java EE сервер (или сервер приложений), внутри него располагаются контейнеры компонентов. В данных контейнерах размещаются (бинго!) компоненты.

Чтобы объяснить архитектуру Java EE приложений, для начала поговорим об уровнях. Какие бывают уровни? Какие Java EE технологии используются на различных уровнях?

Далее мы обсудим, как между собой связаны сервера приложений, контейнеры компонентов и сами компоненты.

Но учтите, что все это — взгляды под различными углами на одно и то же, и очередность тут не так важна.

Уровни приложений

Многоуровневые приложения — это приложения, которые разделены по функциональному принципу на изолированные модули (уровни, слои).
Обычно (в том числе в контексте Java EE разработки) корпоративные приложения делят на три уровня:

- клиентский;
- средний уровень;
- уровень доступа к данным.

1. Клиентский уровень представляет из себя некоторое приложение, которое запрашивает данные у Java EE сервера (среднего уровня). Сервер, в свою очередь, обрабатывает запрос клиента и возвращает ему ответ.
   Клиентским приложением может быть браузер, отдельное приложение (мобильное либо десктопное) или другие серверные приложения без графического интерфейса.

2. Средний уровень подразделяется, в свою очередь, на web-уровень и уровень бизнес-логики.

    a. Web-уровень состоит из некоторых компонент, которые обеспечивают взаимодействие между клиентами и уровнем бизнес-логики.
    На web-уровне используются такие технологии Java EE:
        - JavaServer Faces technology (JSF);
        - Java Server Pages (JSP);
        - Expression Language (EL);
        - Servlets;
        - Contexts and Dependency Injection for Java EE (CDI).

b. Уровень бизнес-логики состоит из компонент, в которых реализована вся бизнес-логика приложения. Бизнес-логика — это код, который обеспечивает функциональность, покрывающую нужды некоторой конкретной бизнес сферы (финансовая индустрия, банковское дело, электронная коммерция). Данный уровень можно считать ядром всей системы.

   Технологии, которые задействованы на данном уровне:
   - Enterprise JavaBeans (EJB);
   - JAX-RS RESTful web services;
   - Java Persistence API entities;
   - Java Message Service.

3. Уровень доступа к данным. Данный уровень иногда называют уровнем корпоративных информационных систем (Enterprise Information Systems, сокращенно —EIS). EIS состоит из различных серверов баз данных, ERP (англ. Enterprise Resource Planning) систем планирования ресурсов предприятия и прочих источников данных. К этому уровню за данными обращается уровень бизнес-логики.

   В данном уровне можно встретить такие технологии, как:
   - Java Database Connectivity API (JDBC);
   - Java Persistence API;
   - Java EE Connector Architecture;
   - Java Transaction API (JTA).

Сервера приложений, контейнеры, компоненты

Взглянем на определение Java EE из Википедии.

Java EE — набор спецификаций и соответствующей документации для языка Java, описывающий архитектуру серверной платформы для задач средних и крупных предприятий.

Чтобы лучше понять, что означает в данном контексте “набор спецификаций”, проведем аналогию с Java-интерфейсом.

Сам по себе Java-интерфейс лишен функциональности. Он просто определяет некоторый контракт, согласно которому реализуется некоторая функциональность.

А вот реализуют интерфейс уже другие классы. Причем у одного интерфейса допустимы несколько реализаций, каждая из которых может друг от друга отличаться некоторыми деталями.

Со спецификацией все точно так же. Голая Java EE — это просто набор спецификаций.

Данные спецификации реализуют различные Java EE сервера.

Java EE сервер — это серверное приложение, которое реализует API-интерфейсы платформы Java EE и предоставляет стандартные службы Java EE. Серверы Java EE иногда называют серверами приложений. Данные сервера могут содержать в себе компоненты приложения, каждый из которых соответствует своему уровню в многоуровневой иерархии. Сервер Java EE предоставляет этим компонентам различные сервисы в форме контейнера.

Контейнеры — это интерфейс между размещенными на них компонентами и низкоуровневыми платформо-независимыми функциональными возможностями, поддерживающими компонент.

Контейнеры предоставляют размещенным на них компонентам определенные службы. Например, управление жизненным циклом разработки, внедрение зависимости, параллельный доступ и т. д. Контейнеры скрывают техническую сложность и повышают мобильность.

В Java EE существует **четыре различные типа контейнеров**:

1. Контейнеры апплетов выполняются большинством браузеров. При разработке апплетов можно сконцентрироваться на визуальной стороне приложения, в то время как контейнер обеспечивает безопасную среду.

2. Контейнер клиентского приложения (ACC) включает набор Java-классов, библиотек и других файлов, необходимых для реализации в приложениях Java SE таких возможностей, как внедрение, управление безопасностью и служба именования.

3. Веб-контейнер предоставляет базовые службы для управления и исполнения веб-компонентов (сервлетов, компонентов EJB Lite, страниц JSP, фильтров, слушателей, страниц JSF и веб-служб). Он отвечает за создание экземпляров, инициализацию и вызов сервлетов, а также поддержку протоколов HTTP и HTTPS. Этот контейнер используется для подачи веб-страниц к клиент-браузерам.

4. EJB (Enterprise Java Bean) контейнер отвечает за управление и исполнение компонентов модели EJB, содержащих уровень бизнес-логики приложения. Он создает новые сущности компонентов EJB, управляет их жизненным циклом и обеспечивает реализацию таких сервисов, как транзакция, безопасность, параллельный доступ, распределение, служба именования либо возможность асинхронного вызова.

Также в Java EE выделяют **четыре типа компонентов**, которые должна поддерживать реализация Java EE спецификации:

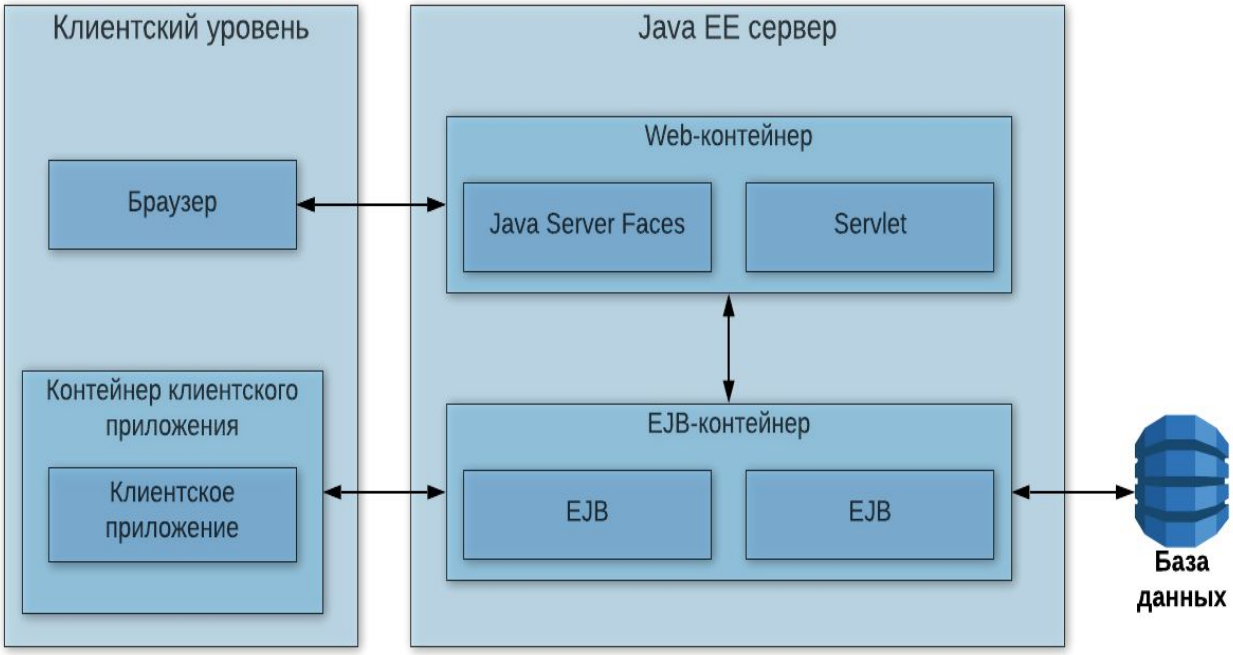
1. Апплеты — это приложения из графического пользовательского интерфейса (GUI), выполняемые в браузере. Они задействуют насыщенный интерфейс Swing API для производства мощных пользовательских интерфейсов.

2. Приложениями называются программы, выполняемые на клиентской стороне. Как правило, они относятся к графическому пользовательскому интерфейсу (GUI) и применяются для пакетной обработки.

3. Веб-приложения (состоят из сервлетов и их фильтров, слушателей веб-событий, страниц JSP и JSF) — выполняются в веб-контейнере и отвечают на запросы HTTP от веб-клиентов. Сервлеты также поддерживают конечные точки веб-служб SOAP и RESTful.

4. Корпоративные приложения (созданные с помощью технологии Enterprise Java Beans, службы сообщений Java Message Service, интерфейса Java API для транзакций, асинхронных вызовов, службы времени) выполняются в контейнере EJB. Управляемые контейнером компоненты EJB служат для обработки транзакционной бизнес-логики. Доступ к ним может быть как локальным, так и удаленным по протоколу RMI (или HTTP для веб-служб SOAP и RESTful).

На диаграмме ниже представлена типичная архитектура Java EE приложения:



Технологии

Итак, с архитектурой разобрались. Общая структура должна быть ясна. В процессе описания компонентов архитектуры мы затронули некоторые технологии Java EE, такие как EJB, JSP и пр. Давайте поближе посмотрим на них.

В таблице ниже приведены технологии, которые используются в основном на клиентском уровне:

Технология	Назначение
Servlets	Java-классы, которые динамически обрабатывают клиентские запросы и формируют ответы (обычно HTML страницы).
Java Server Faces (JSF)	Фреймворк для построения веб приложений с пользовательским интерфейсом. Позволяет включать на страницу компоненты

	пользовательского интерфейса (например, поля и кнопки), преобразовывать и валидировать данные компоненты, а также сохранять эти данные в хранилищах на стороне сервера.
Java Server Faces Facelets technology	Представляет из себя подтип приложения JSF, в котором вместо JSP страниц используются XHTML страницы
Java Server Pages (JSP)	Текстовые документы, которые компилируются в сервлеты. Позволяет добавлять динамический контент на статические страницы (например, HTML-страницы)
Java Server Pages Standard Tag Library (JSTL)	Библиотека тегов, в которой инкапсулирована основная функциональность в контексте JSP страниц.
Expression Language	Набор стандартных тегов, которые используются в JSP и Facelets страницах для доступа к Java EE компонентам.
Contexts and Dependency Injection for Java EE (CDI)	Представляет собой набор сервисов, предоставляемых Java EE контейнерами, для управления жизненным циклом компонентов, а также внедрения компонентов в клиентские объекты безопасным способом.
Java Beans Components	Объекты, которые выступают в роли временного хранилища данных для страниц приложения.

В таблице ниже приведены технологии используемые на уровне бизнес-логики:

Технология	**Назначение**

Enterprise Java Beans (enterprise bean) components	EJB — это управляемые компоненты, в которых заключена основная функциональность приложения.
JAX-RS RESTful web services	Представляет из себя API для разработки веб-сервисов, соответствующих архитектурному стилю REST.
JAX-WS web service endpoints	API для создания и использования веб-сервисов SOAP.
Java Persistence API (JPA) entities	API для доступа к данным в хранилищах данных и преобразования этих данных в объекты языка программирования Java и наоборот.
Java EE managed beans	Управляемые компоненты, которые предоставляют бизнес-логику приложения, но не требуют транзакционных функций или функций безопасности EJB.
Java Message Service	API службы сообщений Java (JMS) — это стандарт обмена сообщениями, который позволяет компонентам приложения Java EE создавать, отправлять, получать и читать сообщения. Что обеспечивает распределенную, надежную и асинхронную связь между компонентами.

В таблице ниже приведены технологии, используемые на уровне доступа к данным:

Технология	**Назначение**
The Java Database Connectivity API	Низкоуровневое API для доступа и

(JDBC)	получения данных из хранилищ данных. Типичное использование JDBC — написание SQL запросов к конкретной базе данных.
The Java Persistence API	API для доступа к данным в хранилищах данных и преобразования этих данных в объекты языка программирования Java и наоборот. Гораздо более высокоуровневое API по сравнению с JDBC. Скрывает всю сложность JDBC от разработчика под капотом.
The Java EE Connector Architecture	API для подключения других корпоративных ресурсов, таких как: ERP (англ. Enterprise Resource Planning, система планирования ресурсов предприятия), CRM (англ. Customer Relationship Management, система управления взаимоотношениями с клиентами).
The Java Transaction API (JTA)	API для определения и управления транзакциями, включая распределенные транзакции, а также транзакции, затрагивающие множество хранилищ данных.

Java EE vs Spring

Конкурентном Java EE считается Spring Framework. Если взглянуть на развитие двух данных платформ, выходит интересная картина. Первые версии Java EE были созданы при участии IBM. Они вышли крутыми, но неповоротливыми, тяжеловесными, неудобными в использовании. Разработчики плевались из-за необходимости поддерживать большое количество конфигурационных файлов и из-за прочих причин, усложняющих разработку.

В то же время на свет появился Spring IoC. Это была маленькая, красивая и приятная в обращении библиотека. В ней также использовался конфигурационный файл, но в отличии от Java EE, он был один. Простота Spring

привела к тому, что практически все стали использовать данный фреймворк в своих проектах.

А далее Spring и Java EE начали свой путь к одному и тому же, но с разных концов. Компания Pivotal Software, разработчик Spring, стали выпускать проект за проектом, чтобы покрыть все возможные и невозможные потребности Java-разработчиков. Постепенно то, что раньше называлось Spring, сначала стало одним из проектов, а потом и вовсе слилось с несколькими другими проектами в Spring Core. Все это привело к неминуемому усложнению Spring по сравнению с тем, каким он был изначально. Со временем следить за всем клубком зависимостей спринга стало уж совсем сложно, и возникла потребность в отдельной библиотеке, которая стала бы загружать и запускать все сама (сейчас где-то икнул так горячо любимый Spring Boot).

Все это время JCP работал над одним — добиться максимального упрощения всего, что только можно внутри Java EE. В итоге в современном EJB для описания бина достаточно указать одну аннотацию над классом, что предоставляет разработчику доступ ко всей мощи технологии Enterprise Java Beans. И подобные упрощения затронули каждую спецификацию внутри Java EE.

В итоге по функционалу Spring и Java EE примерно разделяют паритет. Где-то что-то лучше, где-то что-то хуже, но если смотреть глобально, больших различий нет. То же самое касается сложности работы. И Spring, и Java EE являются превосходными инструментами. Пожалуй, лучшими из того, что сейчас есть, для построения корпоративных сетевых приложений на языке Java.

Однако, Java EE может работать в общем случае только в рамках Enterprise Application Server’a (Tomcat таковым не является), а приложение на Spring стеке может работать на чем угодно (на том же Tomcat), и даже вообще без сервера (так как запустит его внутри себя самостоятельно).

Это делает Spring идеальным инструментом для разработки небольших приложений с GUI на Front-end или для микросервисной архитектуры. Но отказ от зависимости от серверов приложений отрицательно сказался на масштабируемости Spring-приложений.

А Java EE хорошо подходит для реализации масштабируемого монолитного кластерного приложения.

На рынке труда на текущий момент более востребованы разработчики, знакомые со Spring Framework. Так сложилось исторически: в те времена, когда Java EE была излишне усложнена, Spring "набрал клиентскую базу".

И тем не менее, однозначного ответа на вопрос что учить Spring или Java EE, нет. Новичку можно дать следующий совет. Познакомиться (хотя бы поверхностно) с обеими платформами. Написать небольшой домашний проект и на Java EE и на Spring. А далее углубляться в тот фреймворк, который потребуется на работе. В итоге, переключаться между Spring и Java EE не составит большого труда.

Итоги

Масштабную тему, конечно же, одной статьей не охватить! После тонны новых терминов наверняка хочется “приложить” эти знания к жизненному примеру. Поэтому мы продолжим изучать Java EE: уроки практические, по настройке локального окружения для Java EE разработки, вы найдете в следующей статье.