УДК 004-43

Е.П. Сирота

АНАЛИЗ НОВЫХ ПЕРСПЕКТИВ ТЕХНОЛОГИЙ JAVA ENTERPRISE ДЛЯ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ С ПОЯВЛЕНИЕМ POLYGLOT JVM

Аннотация: Рассматриваются архитектуры серверных платформ для корпоративных приложений на языке программирования Java с использованием технологий Java EE, Spring, Akka. Анализируется как при помощи Polyglot JVM возможно сохранить данные архитектуры и технологии и при этом привнести инновации, связанные с появлением новых языков программирования.

Ключевые слова: разработка корпоративных высоконагруженных систем, архитектура серверных платформ для корпоративных систем, Java, Java EE, Spring, AKKA, CORBA, OMA, JVM, Polyglot JVM.

Вступление

Данная статья посвящена анализу архитектур серверных платформ для корпоративных приложений на языке программирования Java с использованием технологий Java EE, Spring, Akka и актуальности использования этих архитектур при смене языка программирования. Последние несколько лет популярен вопрос о том, какой язык заменит язык Java. Ценность использования Java для корпоративных приложений состоит не в самом языке, а в выработанных архитектурах и существующем стеке технологий, которые эффективно решают задачу разработки высоконагруженных корпоративных систем. Эти технологии имеют собственную ценность, отдельную от ценности языка Java.

Возможно ли одновременно заменить язык программирования и при этом сохранить существующие технологии и сделанные инвестиции? Да, возможно.

Данная статья посвящена анализу, как сохранить наработки Java в секторе корпоративных приложений и при этом привнести инновации, связанные с появлением новых языков программирования.

Object Management Architecture

У истоков архитектур серверных платформ для корпоративных приложений находятся архитектура Object Management Architecture (OMA) и спецификация Common Object Request Broker Architecture and Specification (CORBA), разработанные группой Object Management Group (OMG).

Архитектура OMA [1] была разработана для объектных распределенных систем, в которых клиент осуществляет удаленный вызов метода объекта.

Подробнее о типах ресурсов для удаленного вызова указано в [2], раздел 2.1.1.5. В зависимости от того, как трактовать удаленный ресурс,

 ISSN 1560-8956

выделяются следующие ресурсные модели: сервер, сервис, объект. Если внимательно посмотреть на эти типы ресурсов, то для каждого типа “напрашивается” свой тип архитектуры приложения: соответственно клиент-серверная, сервис-ориентированная, распределенная объектная архитектуры.

Итак, OMA создавалась для третьего типа ресурсов – для объектов. Сейчас активнее используются другие типы ресурсов - сервис (например, SOAP), сервер (например, SQL, HTTP).

Но изменение типа ресурса с объекта на сервер/сервис не помешало перенести множество идей из OMA, в частности, на архитектуру Java EE.

Важной частью OMA (рисунок 1) является брокер объектных запросов – Object Request Broker (ORB). ORB – это шире чем просто объектно-ориентированная технология удаленного вызова. Функционально – это частично аналог сервера приложений.

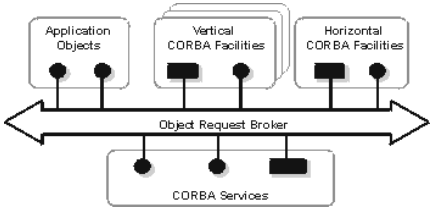

Рис. 1 – Архитектура OMA

CORBA специфицирует, как должен быть реализован ORB.

Архитектура ORB основана на двух важных идеях:

- архитектура клиента должна быть простой;
- все механизмы масштабируемости, которые позволяет серверу выдержать высокую нагрузку, находятся на сервере, в результате чего архитектура сервера получается сложнее.

Именно в ORB по сути вводится понятие контейнера для объектов бизнес-логики (это называется object adapter): объекты бизнес-логики на сервере создает и разрушает контейнер. На рисунке 2 приведена структура ORB и выделен object adapter.

Контейнер был введен с целью обеспечить работу сервера под высокой нагрузкой. Так, 1) клиенту не позволено напрямую создавать объекты на сервере – только через обращение к контейнеру – при этом контейнер гарантирует разрушение объектов после завершения их использования, 2) сервер может применять к объектам механизмы масштабируемости (выгружать объект, если к нему давно не было обращений, загружать, если он понадобился). Все полномочия клиента в данной схеме работы - запросить у сервера ссылку на серверный объект с бизнес-логикой и удаленно вызвать метод объекта.

Кроме контейнера объектов бизнес-логики в OMA введено понятие сервисов (Object Services) как составной части серверной платформы –

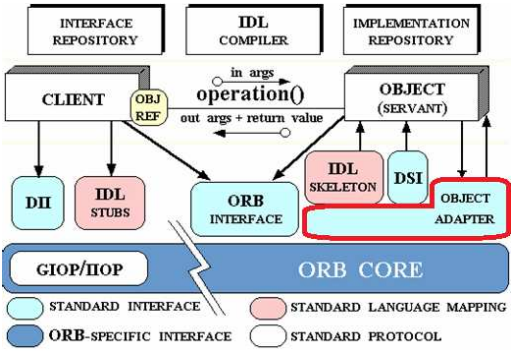

Рис. 2 – Структура ORB, выделен object adapter

см. рисунок 1. Среди сервисов (рис. 3): сервис распределенных транзакций, сервис безопасности, event service и другие (полный список сервисов указан в [3]).

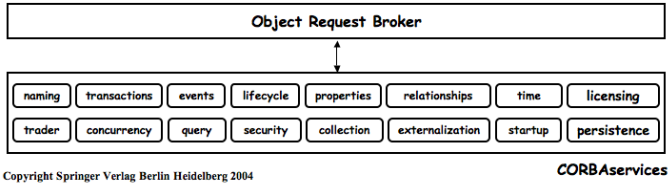

Рис. 3 – CORBA Object Services

Реализация серверного объекта может обращаться к сервисам через упомянутый выше object adapter.

Таким образом, в CORBA заложены идеи, которые активно используются в современных серверных платформах на Java: использование контейнеров и сервисов, как показано на рисунке 4.

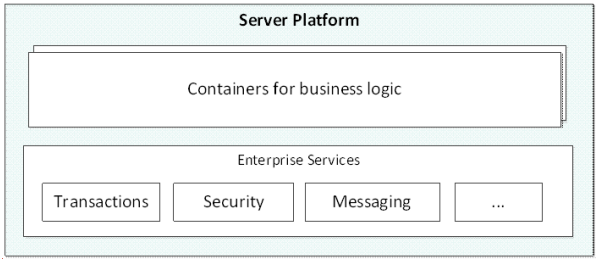

Рис. 4 – Состав серверной платформы: контейнеры и сервисы

 ISSN 1560-8956

В CORBA также было заложено много других идей, которые интенсивно используются в настоящий момент. Так, именно в CORBA было введено понятие “интерсептор” ([4], глава 21). С помощью интерсептора в CORBA могут быть вызваны сервисы во время создания серверного объекта или при вызове серверного объекта. Сейчас интерсепторы активно используется в IoC-контейнерах (IoC - Inversion Of Control) в Spring и в Java EE.

2. Архитектуры платформ Java EE / Spring

Современные архитектуры серверных платформ (в частности, Java EE, Spring) прошли определенную эволюцию после CORBA.

В их состав добавились фреймворки:

- веб-фреймворк;
- IoC-контейнер;
- ORM-фреймворк.

CORBA ориентирована на использование протокола коммуникации клиента с сервером GIOP/IIOP. В настоящее время для коммуникации клиента с сервером используются протоколы HTTP, SOAP и REST (поверх HTTP).

Для обеспечения работы с протоколом HTTP в состав серверной платформы включается сервлет-контейнер. Сервлет-контейнер взял на себя только работу с протоколом HTTP и не представил никакой программной модели для написания бизнес-логики. Код, написанный для сервлет-контейнера, изобилует зависимостями от протокола HTTP.

В связи с этим появились веб-фреймворки, которые выполняют задачи:

1) убирают зависимость кода с бизнес-логикой от протокола коммуникации;

2) вводят архитектурные паттерны для прикладных модулей (например, MVC), которые ускоряют разработку веб-ориентированных приложений;

3) управляют жизненным циклом объектов в областях видимости, имеющих место для веб-приложений.

IoC-контейнер (Inversion of Control) решает следующую задачу: позволяет передать управление жизненным циклом любого объекта на сервере контейнеру, но при этом код бизнес-логики не будет зависеть от самого контейнера.

Важное место в составе платформы занял ORM-фреймворк (Object Relational Mapping) для сохраняемых объектов. Практика показала, что данный фреймворк не должен обладать функциями контейнера (то есть управлять жизненным циклом сохраняемых объектов). Эта ошибка – Container Managed EJB Entity Beans - была допущена в EJB 2.0. Это решение не прижилось на практике.

Типичная архитектура серверной платформы на базе Java представлена на рисунке 5.

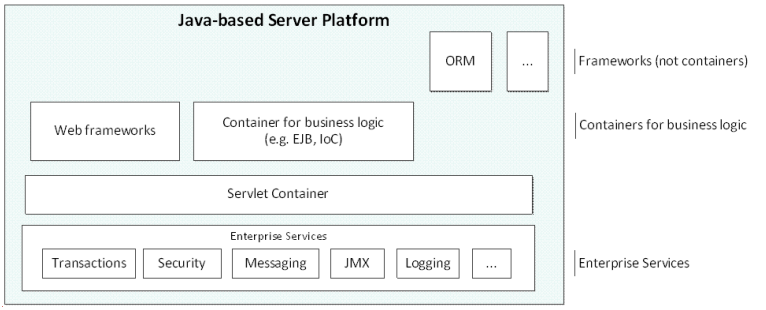

Рис. 5 – Типичная архитектура серверной платформы на базе Java

2.1 Java EE

В разделе, посвященном CORBA, выделены два важных конструктивных элемента, которые интенсивно используются в платформе Java EE – контейнеры серверных объектов и сервисы как составная часть серверной платформы.

Основными контейнерами в Java EE являются сервлет-контейнер, EJB-контейнер и IoC-контейнер, а среди основных сервисов: сервис транзакций JTS, сервис безопасности, JDBC, сервис именований JNDI, сервис сообщений JMS, сервис HTTP, management JMX.

Архитектура платформы Java EE представлена на рисунке 6.

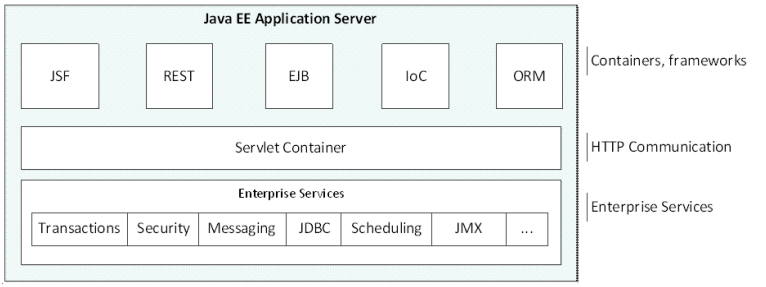

Рис. 6 – Структура платформы Java EE

2.2. Spring

Платформа Spring – представитель той же архитектуры, что и Java EE (CORBA): активно используются контейнеры, сервисы и фреймворки (рис. 7).

Однако есть и различия:

1) Spring может разворачиваться отдельно либо на базе веб-сервера, который предоставляет сервлет-контейнер и базовые сервисы;

 ISSN 1560-8956

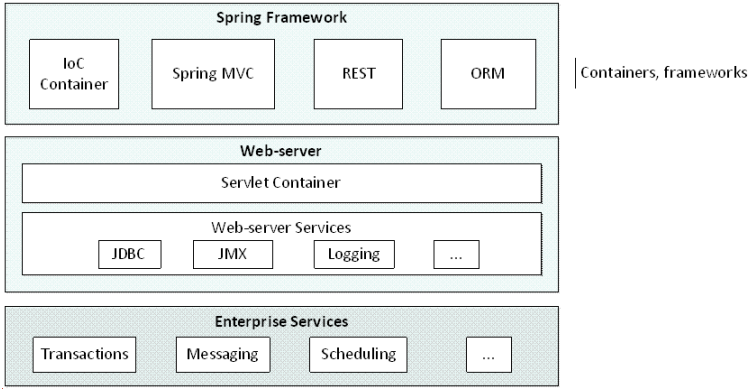

Рис. 7 – Структура платформы Spring

2) Spring поставляет в своем составе API для работы с необходимыми enterprise-сервисами, которые могут быть доставлены по мере использования.

2.3. Spring и Java EE

В данном разделе не производится сравнение этих двух схожих платформ. В данном разделе предлагается то позиционирование, которое было предложено на сессии [5] на конференции JavaOne 2012: Spring – это инновации, Java EE – стандартизация. Инновации в технологиях должны доказать свое право на жизнь, а затем могут быть стандартизированы. Неудачное решение EJB 2.0 Entity Bean до сих пор портит мнение о технологии EJB , причина этой неудачи была в поспешной стандартизации технологии.

3. Java Enterprise без Java?

Описанные выше архитектуры и технологии, которые составляют главную ценность Java в секторе корпоративных приложений, по сути не зависят от Java. Конечно, они работают на платформе Java, но имеют отдельную от Java ценность.

Можно ли сохранить эту ценность и при этом изменить язык программирования? Рассмотрим, почему может понадобиться другой язык программирования.

Суммируя архитектуры из предыдущего раздела, хочу предложить рисунок 8, в который добавлены компоненты бизнес-логики.

Анализируя рисунок 8, возникает следующий вопрос: должны ли компоненты бизнес-логики быть зависимы от языка, на котором реализована серверная платформа.

Если такая независимость возможна, это позволит сохранить все наработки, весь стек технологий серверной платформы, а также привнести инновации и все преимущества других языков, например, функци-

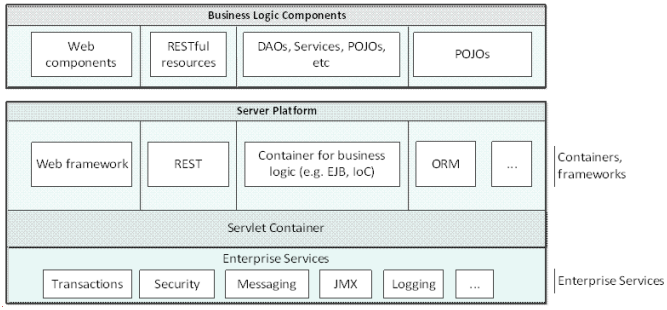

Рис. 8 – Компоненты бизнес-логики

ональных или языков с динамической типизацией. Из преимуществ других языков – лаконичность, замыкания, перегрузка операторов, интуитивный синтаксис для работы с коллекциями, функциональная парадигма программирования. Также динамические языки открывают путь к использованию DSL (Domain Specific Language) с Java EE.

Одновременно возникает вопрос о том, что серверная платформа должна быть открыта для инноваций. Возможно, некоторые из этих инноваций будет удобно сделать на другом языке программирования из-за характера решаемых ими задач.

Перейдем к вопросу как добиться подобной независимости. Независимость серверных объектов от языка реализации серверной платформы была предложена еще в CORBA. Это достигалось стандартизацией платформы и использованием IDL (interface definition language) при написании серверных объектов. Однако жесткая стандартизация серверной платформы закрывает путь инновациям. В результате CORBA мало распространена при построении корпоративных систем.

Для того, чтобы найти решение, необходимо выйти за пределы предыдущего рисунка и поискать фундамент, на котором построена сама серверная платформа. Этот фундамент – виртуальная машина Java (JVM, Java Virtual Machine), см. рисунок 9.

Если серверная платформа и бизнес-логика будут написаны на разных языках, которые выполняются на одной и той же JVM, то это позволит добиться целей:

1) сохранить все существующие наработки серверных платформ;

2) обеспечить открытость серверной платформы для инноваций;

3) разрабатывать бизнес-логику и серверную платформу на разных языках.

Именно поэтому активно развивается направление Polyglot JVM [6].

Итак, использование Polyglot JVM достигает двух целей: открывает путь инновациям и позволяет сохранить все наработки в секторе корпоративных приложений.

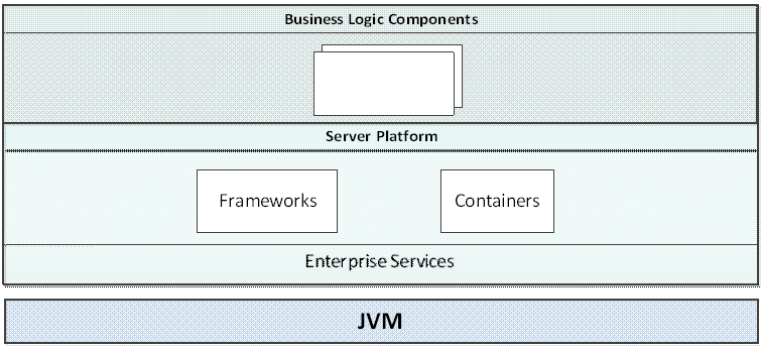

Рис. 9 – Виртуальная машина Java (JVM, Java Virtual Machine)

4. Polyglot JVM

JVM всегда была отделена от Java и выполняла собственный набор инструкций – байткод. Программа, написанная на другом языке и скомпилированная в байткод, могла быть выполнена на JVM.

Однако до JDK 6 и JDK 7 это было возможно только для языков со статической типизацией.

Scala – язык со статической типизацией, появившийся в 2003 году – без проблем компилировался в байткод на основе существующего на то время набора инструкций для JVM.

Первый шаг для поддержки языков с динамической типизацией был сделан в JDK 6 - была добавлена библиотека Java Scripting API (javax.script.*). Однако это решение было частичным. Для того, чтобы скриптовый код работал, необходим был scripting engine для этого языка. Такое решение не отличалось производительностью, так как по сути выполнялась “двойная” интерпретация – сначала интерпретация скриптового кода с помощью engine-а, а потом интерпретация кода engine виртуальной машиной.

Настоящим “полиглотом” JVM стала в JDK 7 благодаря появлению новой инструкции для JVM invokedynamic, специально предназначенной для динамических языков. Благодаря этой инструкции производительность выполнения скриптового кода значительно улучшена.

На данный момент поверх JVM могут работать следующие языки: Java, Scala, JRuby, Groovy, Clojure, Jython и другие.

5. Альтернативные архитектуры и Polyglot JVM

Сейчас развиваются и другие серверные платформы, например, на основе event driven architecture (EDA), в частности, Akka [7].

Для платформ на основе EDA также применима идеология контейнеров и сервисов. Однако список сервисов для них должен быть пересмотрен и отличаться от списка сервисов CORBA/Java EE/Spring. Это

красиво демонстрируется на примере сервиса распределенных транзакций. Использование событий вместо удаленных вызовов (RPC) позволяет добиться слабой связанности. Однако при таком подходе становятся непонятно, где устанавливать границы транзакции. Установка границ транзакции ведет к сильной связанности.

Таким образом, для EDA та же картина – наработки в области EDA имеют собственную ценность, и не зависят от языка программирования.

Традиционно Akka рассматривается в паре со Scala или Java. Однако есть прецеденты использования других языков, например Akka + Groovy.

Использование Polyglot JVM для EDA также позволяет сохранить наработки в технологиях и платформах и при этом использовать преимущества других языков.

Выводы

В статье рассмотрены архитектуры и технологии серверных платформ, используемые при разработке корпоративных приложений на языке Java. Эти наработки имеют собственную ценность, отличную от ценности языка Java. Благодаря Polyglot JVM можно сохранить все эти технологии и при этом открывается возможность использования новых языков при разработке корпоративных приложений.

Итак, Polyglot JVM открывает путь инновациям и позволяет сохранить все наработки в секторе корпоративных приложений:

1) сохранить все существующие наработки серверных платформ для Java;

2) обеспечить открытость серверной платформы для инноваций;

3) разрабатывать бизнес-логику на языке, отличном от языка реализации серверной платформы, и привнести в разработку бизнес-логики все преимущества функциональных языков и языков с динамической типизацией;

4) использовать те языки, которые знает команда и не тратить время и деньги на переучивание.

Дальнейшие исследования перспективно проводить в следующих направлениях:

1) выделение классов задач при разработке корпоративных приложений, которые эффективнее разрабатывать на языках, отличных от Java;

2) развитие архитектур серверных платформ и разработка новых сервисов в их составе на языках, отличных от Java.

Литература

1. Object Management Architecture Resource Page. - http://www.omg.org/oma
2. DCE 1.1: Remote Procedure Call. - https://www2.opengroup.org/ogsys/catalog/c706

3. OMG Specifications, CORBAservices Specifications. - http://www.omg.org/spec/#CS

4. The Common Object Request Broker: Architecture and Specification, Revision 2.3 - http://www.omg.org/spec/CORBA/2.3/

5. R.Hightower, B.Ertman, G.Dickens, Ch.Beams, A.Gupta. Java EE and Spring Framework Panel Discussion, Java One 2012 Conference, San-Francisco 2012. - https://oracleus.activeevents.com/connect/sessionDetail.ww ?SESSION_ID=6430

6. B.Evans, M.Verburg, Polygloth Programming on the JVM. - Java Magazine, Nov-Dec. 2011. – p.50-55. - http://www.oraclejavamagazine-digital.com/javamagazine/20111112#pg51

7. T.Neward. Building Actor-Based Systems Using Akka Framework. - Java Magazine, Nov-Dec. 2012. – p. 55-59. - http://www.oraclejavamagazine-digital.com/javamagazine/20121112?pg=56#pg56

Отримано 20.11.2012 р.