**Сравнение стеков Spring и Java EE: возможности и ограничения**

**2017**

Практически на каждой конференции, где я выступаю по связанным с Java вопросам, находится собеседник (точнее всегда много), для которого Java = Spring, без вариантов. Попытки рассказать людям, что в мире есть множество других технологий, вызывают неподдельное удивление в их глазах.

У меня же на фирме вследствие определенных ограничений (мы работаем на внутренний рынок и поэтому вынуждены держать рейты невозможно низко для аутсорса) мы стараемся оптимизировать расходы клиентов на разработку. И разработка на стеке Java EE оказалась существенно дешевле разработки на стеке Spring при некоторых ограничениях.

Сразу хочу оговориться: я не рассматриваю холиварного противостояния — что на самом деле «тру», а что нет, — не разбираю вопросов «правильности» или соответствия идеалам красоты. Только деньги, ничего личного. Я же все-таки уже не разработчик, а «бездушный галеровладелец». Как пишут на одном широко известном ресурсе :)

Начнем с истории вопроса

Я — человек занудный. Поэтому не могу отойти от канонов академического образования: для любой вещи сначала надо сказать о том, откуда она вообще появилась на свет.

Итак, сначала было слово... А, нет, сначала была Java. И был Java Community Process (JCP), процедура в соответствии с которой и происходят все правки и выходят новые версии самой Java и связанные с ней спецификации. Например — Java EE и ее составные части.

И надо сказать, JCP — еще та неповоротливая и забюрократизированная система. Да еще и построенная на демократических принципах: индивидуально в JCP может войти любой и бесплатно. Фирмами — за денежку. И голосовать по поводу всех важных и не очень решений, связанных с Java. Короче — такая международная Верховная Рада. Только про Java. Ну вы поняли.

Тут я должен сделать еще одно отступление (люблю их!) и напомнить, что Java — первопроходец в массе разных направлений. Поэтому сейчас, конечно, приятно ругать JCP за кучу принятых ошибочных решений. Проблема в том, что на тот момент просто не было опыта таких решений, и кто-то же должен был быть первым. Примером такого плохого решения как раз можно считать спецификации EJB 1.0 и 1.1, о которых в Wikipedia сказано мрачно: «The EJB specification was originally developed in 1997 by IBM and later adopted by Sun Microsystems (EJB 1.0 and 1.1) in 1999». Короче, ребята из JCP вообще не знали, как делать большие Enterprise проекты (и никто на тот момент как следует не знал!). Но вот ребята из IBM (между прочим — один из исполнительного комитета JCP) пришли и сказали: «Мы знаем, как это делать». И в непередаваемом IBM стиле наваяли спецификацию. Два года, потраченные на адаптацию этого решения намекают, что документация была еще та.

В итоге у Java EE появился ключевой его элемент — EJB. Я как сертифицированный специалист по этим первым EJB могу точно сказать: это было чудовищно сложно и криво. Однако у EJB были свои преимущества: автоматическая, прямо из коробки, масштабируемость путем поддержки кластеров серверов приложений, опять же из коробки поддержка авторизации и самое главное — это был стандарт!

Итогом появления первых EJB стали две вещи. Во-первых, страшный хайп в java-мире: «Вот сейчас мы завоюем мир. У нас есть мощные EJB!» Честно-честно, в те годы в мире Java было полно хайпа. Вообще хочу заметить, что если быть в IT больше 10 лет (а лучше 20), то можно увидеть, что хайпы повторяются с определенной цикличностью. Например, современный хайп по Front-end — уже третий. И я точно знаю, чем он закончится (спойлер — возвратом к ультратонкому клиенту). Если интересно, чтобы я рассказал об этом в отдельной статье — напишите здесь в комментариях.

Но я отвлекся. Вторая вещь, которая стала следствием первой — это жуткое разочарование в EJB у всех, кто с ними работал. Реально, первая версия была чудовищно неудобной, глючной, требующей дикого количества дополнительных настроечных файлов к каждому бину и странных действий от программиста. А самое главное — инструментарий первых EJB оставлял желать смерти авторам. Понять по стек-трейсам, что именно у вас не так, было крайне сложно.

В итоге все те, кто повелся на рекламу и хайп, начали от новой технологии плеваться и поносить ее всеми ругательными словами, которые только можно придумать. Достаточно сказать, что термин POJO был придуман именно как альтернатива громоздким, корявым и неудобным конструкциям первых библиотек Java EE и в особенности EJB.

Но как говорят, свято место пусто не бывает, и программистам все равно нужны были фреймворки для облегчения работы. И они, естественно, пришли. Были их сотни, большинство так и сгинуло, но несколько фреймворков выжило. И парочка из них определила все дальнейшее развитие Java. Я имею в виду Hibernate и Spring DI aka IoC.

История Hibernate не менее драматична, чем история со Spring, но ее я расскажу как-нибудь в другой раз, если вам будет интересно.

А сейчас разговор про Spring. Итак, Spring был маленькой, красивой и приятной в обращении библиотекой. Да, еще не было никаких аннотаций (их добавили позже — уже в Java 5), все конфигурирование заключалось в правке XML файла. Но он был один! Он имел простую и понятную структуру а-ля web.xml! И по сравнению с ужасом конфигурационных файлов Java EE того времени был просто лапочкой. О чем там говорить, все, включая меня, были в диком восторге от Spring. Прошло совсем немного времени, и наличие Spring на проекте стало просто обязательным.

Надо сказать, что JCP не стали ревновать выскочек к успеху и самоотверженно начали впиливать поддержку Spring IoC в стек Java EE. И всё чудесно вертелось. Все, кто хотели (скажем честно — вообще все), использовали Spring IoC в своих проектах, не обращая внимания на то, на чем проект написан.

Но время шло, и жизнь не стояла на месте. Компания Pivotal, разработчик Spring, решила захватить мир, в смысле... Ну да, захватить мир, а что — так нельзя было? И начала клепать новые проекты со словом Spring в названии, чтобы покрыть все возможные и невозможные потребности Java разработчиков. Постепенно то, что раньше называлось Spring, сначала стало одним из проектов, а потом и вовсе слепилось с несколькими другими проектами в то, что сейчас называется Spring Core. А список всех остальных проектов (который раньше висел на сайте в виде красивой инфографики) спрятали от посторонних лиц. И правильно. Если все эти проекты вынести на инфографику, то придется использовать 6-й шрифт, и то картинка будет во всю стену. Постепенно следить за всем этим адом из зависимостей стало уж совсем сложно и появилась необходимость в отдельной библиотеке, которая должна сама все загружать и запускать. **Ага, привет, Spring Boot.**

Что же случилось с забытой всеми J2EE? Ее переименовали в красивую Java EE, убрав пугающую новичков двойку. Все это время JCP работал над одним — добиться максимального упрощения всего, что только можно. В итоге в современном EJB для описания бина достаточно указать одну аннотацию над классом. Все, у вас уже есть доступ ко всей мощи EJB.

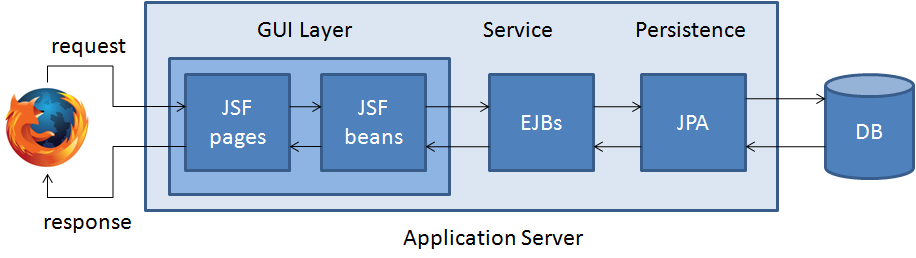
И что? Обрадованные Java разработчики повыкидывали ставший монстром Spring и вернулись к Java EE, который стал таким простеньким? А фиг там. Мыши плакали, кололись, но продолжали жрать кактус.

Собственно, для того, чтобы обратить внимание «мышей» на колючки я и написал эту статью.

Но вступление, кажется, затянулось. Поехали ближе к теме, рассмотрим оба стека технологий в реальном использовании. Я уже говорил, что за последний год управлял несколькими проектами на Spring стеке и одним на Java EE стеке. Так что есть с чем сравнить.

Стек Java EE

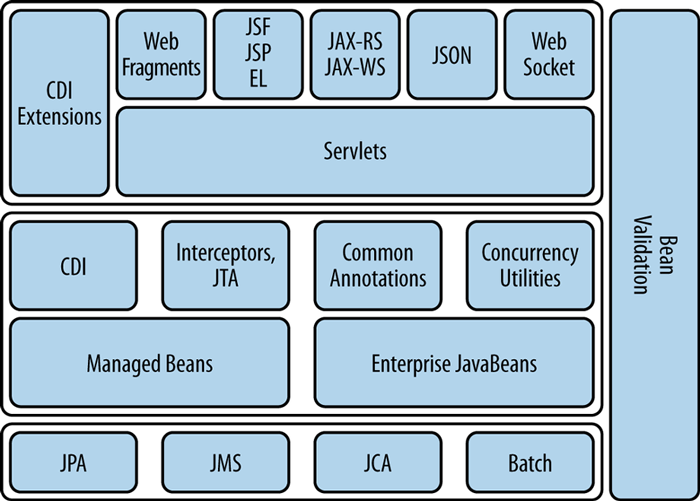
Для начала вот вам картинка с простейшим Java EE приложением.



Как видите, ничего сложного вообще. Запрос обрабатывается JSF страницей, а затем — бином. Управление дальше уходит на бизнес-логику в EJB, которая работает с базой данных через JPA. Пока все просто и понятно. Самое приятное во всей этой картине, что если нагрузка на это приложение увеличится на несколько порядков, то его схема не изменится вообще никак. Приложение надо будет установить просто на кластер из нескольких мощных нод, и все будет работать в кластерном окружении без малейших правок. Если мощности все равно будет не хватать, то достаточно добавить еще нод в кластер — хватит.

Именно из-за этого решения на стеке Java EE я рекомендую использовать в тех случаях, когда вопрос масштабирования для вашего приложения самый критический. Но на этом я остановлюсь потом еще раз подробнее.

Но это мы изучали самое простое Java EE приложение. Посмотрим на что-то более мощное.

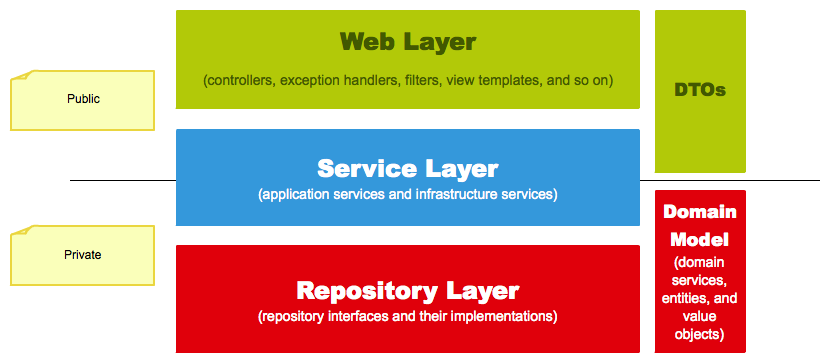


Вот так выглядит Java EE с точки зрения его документации. Опять же — скажем, не Rocket Science. Однако все необходимо есть, более того — есть из коробки в любом Enterprise Application Server. Причем все это работает из коробки вместе и на распределенном (кластерном) окружении так же.

Пока просто запомним это и двинемся вперед.

Стек Spring

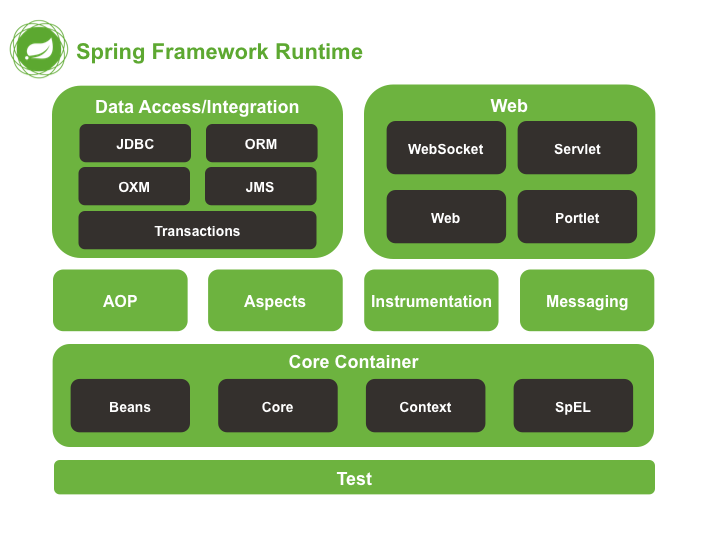
Простое приложение на Spring стеке не сильно-то и отличается от Java EE.



Картинка отличается от предыдущей по большому счету только большей детализацией. Так что среднее приложение на Spring стеке от приложения Java EE не отличается практически никак.

Единственное отличие, которое сразу стоит озвучить, — это то, что приложение на Java EE может работать в общем случае только в рамках Enterprise Application Server’a (напоминаю, что Tomcat им 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 стек идеальным для реализации элементов микросервисной архитектуры, но за счет отказа от поддержки всех возможностей серверов приложений. В общем случае кластерное приложение — это не про Spring, и вопросы масштабирования для Spring приложения должны решаться отдельно. В большинстве случаев — существенно затратнее.

Посмотрим на весь стек библиотек Spring. К сожалению, как я уже говорил, сейчас библиотек чересчур много, так что все их выносить на одну картинку они постеснялись. На сайте до сих пор красуется вот эта картинка примерно пятилетней давности.



Если присмотреться, то все выглядит весьма похоже на предыдущую картинку :) Да так оно на самом деле и есть. У всех выживших на сегодняшний момент стеков (двух) есть практически идентичное представление о том, как надо писать Enterprise приложения.

Каждый стек считает, что объекты предметной области должны быть тупыми Value Object, а сама логика бизнес-слоя должна храниться отдельно от них (в сервисах у Spring и в EJB у Java EE). А это, как вы понимаете, — разделение данных и поведение — ни разу не соответствует ООП. Это самый что ни на есть обычный процедурный код со всеми своими проблемами. Как минимум мы не можем использовать ни единого шаблона проектирования. Как будто и не было всех этих лет развития Computer Science.

Да, это тема для отдельного разговора. Если интересно — пишите в комментариях, может, я созрею написать статью и про это.

Попарное сравнение стеков

Сравнение стеков было бы неполным, если бы мы не обсудили, что хорошего и что плохого в каждом из них. Примерно так и сравнивают автомобили: этот быстрее разгоняется, а у этого есть дополнительная розетка на втором ряду. Тот, кому важнее быстрый разгон, возьмет первый, а кому розетка — второй.

Итак, поехали.

**Spring DI (Dependency Injection) vs CDI (Context Dependency Injection)**

Даже из названия понятно, что ядра в обоих стеках совершенно идентичны по характеристикам. Так и есть. Фактически в Java EE стеке изначально и использовался спринговый DI. Но после того как его включили прямо в Core, использовать его отдельно стало невозможно. Пришлось делать свой DI с аннотациями и программистками.

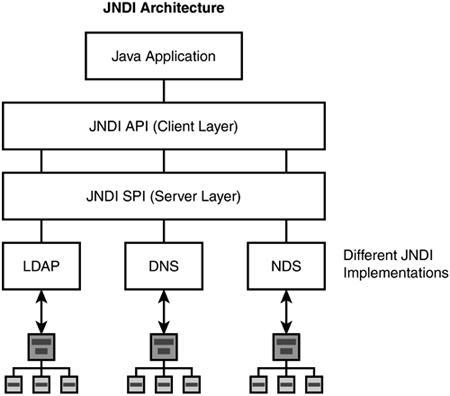
Получилось в общем-то неплохо, даже чуть-чуть лучше, на мой вкус. Но различия настолько незначительны, что ими можно пренебречь.

**Spring Beans vs EJB (Enterprise Java Beans)**

Тут, скажем, ситуация прямо противоположная. Spring beans — это просто обычные джава бины, которые можно куда-то заинжектить и ничего больше. EJB — достаточно мощная штуковина, в которую встроена поддержка распределенного (мультинодного) исполнения, включая распределенный сборщик мусора, аутентификацию, поддержку транзакций и черти что еще. Другой разговор, если все это вам не надо... Ну вы поняли :)

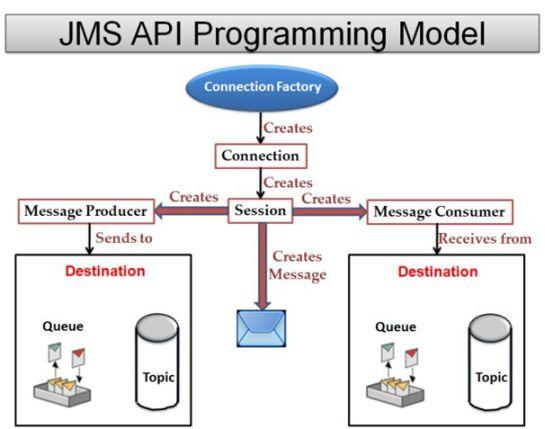
**Spring Service Locator vs JNDI**

Тут тоже ситуация аналогичная. Служба, помогающая найти в том же JVM сервис, и служба, которая может работать на распределенной системе и находить соответствие между чем угодно. Включая привязку к LDAP компании... Формально, предназначены они для одного и того же, однако согласитесь — разница на лицо.



**@Async vs JMS**

Асинхронность также реализована в обоих стеках на разном уровне. В стеке Java EE это отдельная, да, вы правильно догадались, распределенная служба, поддерживающая передачу сообщений не только точка-точка, но и многим получателям. Естественно, отказоустойчивая, персистентная и так далее. В общем, ситуация аналогичная с предыдущими пунктами. Простое и быстрое против распределенной навороченной фиговины.



**Spring MVC vs Java Server Faces (JSF)**

Проблема этого сравнения в том, что SpringMVC в качестве темплейтного движка обычно используется **Thymeleaf** или в более современном варианте — какой-либо Front-end движок (**React, Angular** & etc). В то время, как JSF — полноценное GUI решение с поддержкой AJAX из коробки и даже SPA приложений с помощью дополнительных библиотек, включая такие гиперудобные и красивые, как PrimeFaces и IceFaces.

Вот тут я вынужден обратиться к моему личному опыту и рассказать про несколько наших проектов.

Первый проект решили стартовать на стеке Spring. ТЗ было достаточно простое, куча бизнес-логики, но интерфейс достаточно простой и особых проблем вызывать не должен был. И надо сказать, поначалу все шло отлично — MVP мы нарисовали с опережением графика месяца на полтора. Заказчик сказал: «Вау, ребята, вы — молодцы. Делаем теперь основной проект. Мы же можем задействовать MVP?» Мы его заверили, что ни строчки кода не пропадет. Он сказал: «Отлично, давайте начнем с того, что добавим во все формочки AJAX». Все-таки сейчас 2017 год, перегрузка страниц для заполнения формы — как-то не очень.

**И тут оказалось, что нативной поддержки AJAX в Spring MVC + Thymeleaf просто нет.** Надо каждую операцию писать руками. А у нас практически каждый элемент требует AJAX поведения, причем в разных ситуациях — разного. И писать эту тонну JavaScript мои джаверы вообще не горят желанием. Ну что же. Скрепя сердце были вынуждены выкинуть весь написанный интерфейс **и переделать все приложение на Angular** (для чего взять еще и Front-end разработчика). И заняло это переписывание где-то больше месяца. Вот и считайте, в какие деньги вышло заказчику наше решение использовать стек Spring.

Второй проект был практически идентичен первому, но в нем заказчик изначально просил учесть серьезные нагрузки в будущем — надо мол будет обрабатывать миллионы обращений. Все обдумав, я решил делать проект на стеке Java EE. Думаю, сложнее морду рисовать, зато в будущем будет легкость масштабирования. И что бы вы думали? Мои разработчики нарисовали примерно аналогичное по объему MVP еще быстрее, но при этом сразу из коробки с поддержкой AJAX. Продолжают работать только джаверы, фронтендер им не нужен. Да, конечно, рано или поздно надо будет сделать красивый дизайн, но дизайнер парт-тайм — это не фронтендер фулл-тайм по деньгам.

**Spring Data vs JPA**

Это единственный пункт, в котором Spring кроет как бык овцу стек Java EE. Spring Data — прекрасна, великолепна, быстра и удобна. Снимаю шляпу. Очень жаль, что аналога в Java EE стеке нет...

**Spring Security vs JAAS**

Что хотел я сказать... Spring Security — фреймворк, который мне создал на моих проектах проблем больше, чем все остальные вместе взятые. Возможно, документация фиговая (это единственный Spring фреймворк с фиговой документацией). Возможно, сама тема сложная, но мои программисты традиционно мучаются с ней очень много. Полномасштабного использования JAAS у нас пока не было, но то, что простейшее — прилепляется легко.

Отличия фреймворков вы понимаете и сами. JAAS, естественно, поддерживает распределенность.

**Spring Boot vs Enterprise Server**

Все фанаты спрингового стека любят превозносить Spring Boot со словами «он прекрасен, с ним так все легко», забывая, что этот фреймворк — настоящие костыли, без которых связать воедино весь тот ад, который нафигачили ребята из проекта Spring просто очень сложно.

Напоминаю, что аналогом Spring Boot для стека Java EE является просто любой Enterprise server. Там уже есть все нужные фреймворки нужных и консистентных версий, уже настроенные для работы друг с другом.

Выводы

* Не надо пытаться все приложения делать на каком-то одном стеке — они предназначены немного для разных вещей. Но надо четко понимать, для чего каждый из стеков подходит лучше. Проще говоря, Java EE — для легко масштабируемого монолитного приложения, Spring — для совсем маленьких приложений с GUI на Front-end или для микросервисной архитектуры.
* Не надо забывать и о том, AJAX — это сейчас очень важно. Если вы планируете делать серьезное приложение на SpringMVC + Thymeleaf — готовьтесь серьезно терять во времени разработки. Писать AJAX руками — это больно. Так что либо React/Angular (модно, красиво, молодежно!) либо по старинке JSF + PrimeFaces. Не настолько прекрасно, но тоже весьма ничего. И без Front-ender. Заметим, что именно для приложений бек-офиса (всяких админок и т. п.) PrimeFaces и их аналоги предлагают запредельное количество готовых бизнес-компонентов, типа календарей, колор-пикеров, часовых поясов и всякого такого. Скорость разработки вы существенно повысите
* Ваши программисты могут и не знать стека Java EE. Как показала моя практика — учатся они ему очень быстро.
* И последнее. Я думаю, маятник, который когда-то вынес Spring на вершину, может пойти обратно к Java EE и скинуть Spring вниз. Я бы не исключал такой возможности.