1 Spring в реальном мире

В этой главе

- ✓ Что такое фреймворк.
- ✓ Когда нужно, а когда не следует использовать фреймворки.
- ✓ Что такое фреймворк Spring.
- ✓ Как применять Spring в реальных условиях.

Фреймворк Spring (или просто Spring) — это прикладной фреймворк, который является частью экосистемы Java. *Прикладной фреймворк* — это пакет типичных функций программного обеспечения, образующих базовую структуру для разработки приложения. Прикладной фреймворк позволяет тратить меньше усилий при написании приложения, так как не приходится создавать весь код программы с нуля.

В настоящее время Spring используется для написания самых разных программ, от крупных серверных решений до средств автоматизации тестирования. Согласно многочисленным отчетам об исследованиях в области Java-технологий, таких как JRebel 2020 года или JAXEnter, Spring в настоящее время является наиболее востребованным фреймворком.

Spring популярен: разработчики стали чаще использовать его не только с Java, но и с другими JVM-языками. В последние несколько лет наблюдается впечатляющее увеличение числа разработчиков, применяющих Spring с Kotlin (еще одним популярным языком семейства JVM). В этой книге мы сосредоточимся на основах Spring: я познакомлю вас с важнейшими аспектами его использования

на примерах реальных приложений. Чтобы вам было удобнее и вы могли лучше сфокусироваться на Spring, я буду использовать только примеры на Java. В издании вы изучите и отработаете на практике такие основные приемы, как подключение к базе данных, установка соединения между приложениями, обеспечение безопасности и тестирование приложений.

Прежде чем перейти к следующим главам и углубиться в технические детали, поговорим о фреймворке Spring в целом и о том, где его стоит применять. Почему Spring столь популярен и когда им следует пользоваться?

В данной главе на примере Spring мы подробно рассмотрим, что такое фреймворк вообще. В разделе 1.1 мы обсудим преимущества использования фреймворков. В разделе 1.2 вы познакомитесь с экосистемой Spring и с теми ее компонентами, которые понадобятся для начала работы с ней. Затем я расскажу вам о возможных областях применения фреймворка Spring и о реальных сценариях из практики, раскрытых, в частности, в разделе 1.3. В разделе 1.4 мы обсудим случаи, когда использование фреймворка может оказаться неудачным решением. Все эти моменты следует прояснить прежде, чем вы начнете использовать Spring, — чтобы потом не забивать гвозди микроскопом.

Возможно, в зависимости от ваших исходных знаний, эта глава покажется вам сложной. В ней представлены понятия, с которыми вы, вероятно, еще не знакомы — и это может вызывать беспокойство. Но не волнуйтесь — даже если вы пока какие-то моменты не поймете, в процессе изучения книги они прояснятся. Иногда в процессе изложения я буду ссылаться на что-то описанное в предыдущих главах. Я буду так делать, поскольку изучение такого фреймворка, как Spring, не проходит линейно. Иногда придется подождать, пока соберется еще несколько кусочков пазла и станет видна вся картина. Но в итоге вы получите ясное представление о Spring и приобретете ценные навыки, необходимые для профессиональной разработки приложений.

1.1. ЗАЧЕМ НУЖНЫ ФРЕЙМВОРКИ

Чтобы у вас появилось желание что-то использовать, вам нужно знать, чем это «что-то» может быть вам полезно. Это касается и Spring. В данном разделе мы поговорим о фреймворках. Что это такое? Как и зачем появилась концепция фреймворков? Я познакомлю вас с этими важными моментами, для чего поделюсь собственными накопленными знаниями и покажу, как использовать различные фреймворки, включая Spring, при решении практических задач.

Прикладной фреймворк — это набор функционала, на базе которого строятся приложения. Прикладной фреймворк предоставляет широкий набор инструментов и функций, которые можно применять в разработке. Вы не обязаны использовать их все. В зависимости от требований, предъявляемых к разрабатываемому приложению, вы будете выбирать ту часть фреймворка, которая вам нужна.

В отношении прикладных фреймворков мне нравится такая аналогия. Приходилось ли вам покупать мебель, скажем, в «Икее»? Предположим, вы решили приобрести платяной шкаф. Но вы получите не готовый шкаф, а набор деталей для его сборки и инструкцию по сборке. А теперь представьте, что вы заказали шкаф, но вместо необходимых частей вам выдали вообще все, из чего можно собрать любую мебель: стол, шкаф и т. п. Если вам нужен только шкаф, то придется найти в этой куче нужное и собрать его. Именно так работает прикладной фреймворк: он предлагает все возможные компоненты программного обеспечения, которые могут понадобиться при создании любого приложения. Вам необходимо знать, какие функции выбрать и как их собрать, чтобы получить правильный результат (рис. 1.1).

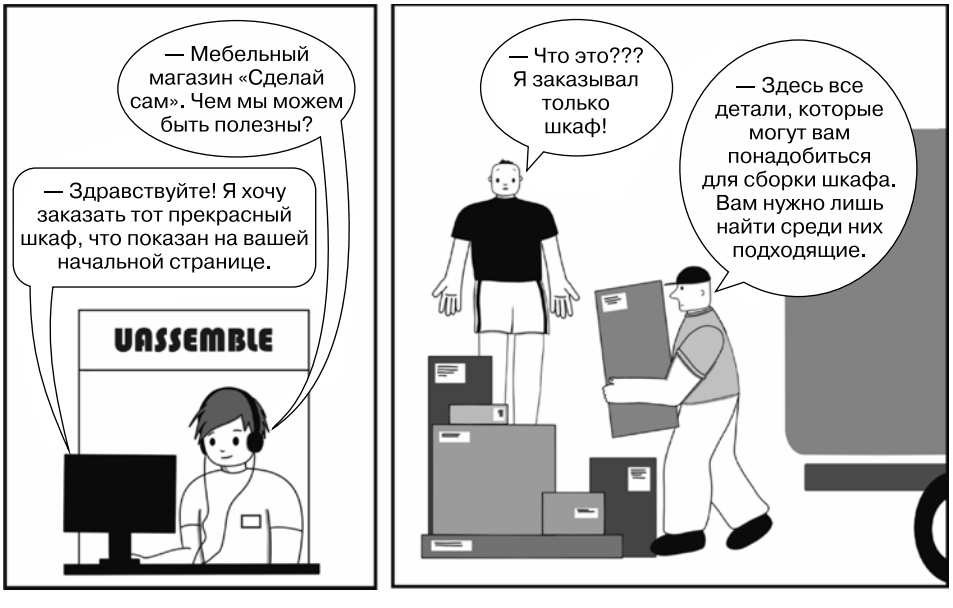

Рис. 1.1. Дэвид заказал шкаф в магазине «Сделай сам». Но магазин (фреймворк) доставил Дэвиду (программисту) не только те детали (компоненты программного обеспечения), которые нужны для сборки его шкафа (приложения), а вообще все возможные детали, которые могут понадобиться при сборке любого шкафа. И Дэвиду (программисту) придется самому решать, какие детали (программные компоненты) необходимы и как их собрать, чтобы получить желаемый результат (приложение)

Концепция фреймворка не нова. На протяжении всей истории разработки ПО программисты замечали, что многие фрагменты написанного ими кода можно использовать повторно в других приложениях. Поначалу, когда приложений было мало, каждое из них было уникально и писалось с нуля на том или ином языке программирования. По мере того как область применения программного

обеспечения расширялась и на рынке появлялось все больше продуктов, стало еще заметнее, как много приложений имеют сходные требования. Вот некоторые из них:

- в каждом приложении предусмотрены уведомления при ошибках входа, предупреждения и информационные сообщения;
- в большинстве приложений для обработки изменений данных используются транзакции. Транзакции являются важным механизмом, который обеспечивает целостность данных. Подробнее мы рассмотрим эту тему в главе 13;
- в большинстве приложений применяются механизмы защиты от одних и тех же распространенных уязвимостей;
- в большинстве приложений используются одни и те же механизмы обмена данными с другими приложениями;
- в большинстве приложений есть одни и те же механизмы повышения производительности, такие как кеширование и сжатие данных.

Данный список можно продолжать. В сущности, код бизнес-логики, реализованной в приложении, значительно меньше, чем вся эта внутренняя начинка (которую еще называют «сантехникой»).

Говоря о «коде бизнес-логики», я имею в виду код, реализующий бизнес-требования приложения. Именно он реагирует на ожидания пользователя от продукта. Например, пользователю нужно, чтобы при щелчке кнопкой мыши на определенной ссылке был сформирован отчет. Эту функциональность реализует часть кода создаваемого вами приложения — и именно эту часть разработчики называют кодом бизнес-логики. Однако любое приложение делает и ряд других вещей: обеспечивает безопасность, журналирование, целостность данных и т. п. (рис. 1.2).

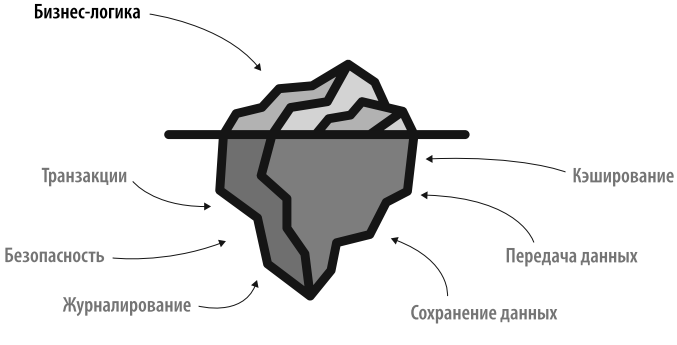

Рис. 1.2. Взгляду пользователя доступна только верхушка айсберга. Пользователи видят главным образом результат работы бизнес-логики. Но это лишь малая доля того, что представляет собой полный функционал приложения. Подобно айсбергу, большая часть которого находится под водой и скрыта от глаз, мы не видим основную часть кода корпоративного приложения, поскольку она выполняется посредством зависимостей

Более того, именно бизнес-логика отличает одно приложение от другого с точки зрения функционала.

Если взять два приложения разных типов — например, систему поиска попутчиков и социальную сеть, — то увидим, что сценарии использования у них разные.

ПРИМЕЧАНИЕ

Сценарий использования (use case) — это причина, по которой человек использует приложение. Например, в приложении поиска попутчиков сценарием использования будет «найти машину». А для приложения доставки еды — «заказать пиццу».

Вы можете выполнять различные действия, но в обоих случаях они будут требовать сохранения данных, их передачи, журналирования, настройки параметров безопасности, возможно, кеширования и т. п. Функционал, не относящийся к бизнес-логике, может многократно внедряться в разные приложения. Имеет ли смысл каждый раз заново его переписывать? Разумеется, нет:

- ведь, используя готовые модули вместо их разработки с нуля, вы не тратите время и деньги зря;
- в готовом решении, применяемом во многих приложениях, вероятность ошибок меньше, так как его уже протестировали другие люди;
- поскольку многие разработчики уже освоили этот функционал, вы можете пользоваться советами сообщества. Если бы вы реализовали собственный код, в нем разбиралось бы гораздо меньше людей.

ИСТОРИЯ ПЕРЕХОДА

Одним из первых продуктов, над которыми я работал, была огромная система, написанная на Java. Она состояла из нескольких приложений, построенных на базе сервера с устаревшей архитектурой. Все они были написаны с нуля на Java SE. Разработка приложения на данном языке началась 25 лет назад — главным образом поэтому оно было таким, каким было. На момент его создания никто не мог представить, до каких размеров оно однажды дорастет. Тогда еще не существовало более совершенных концепций системной архитектуры; из-за низкой скорости интернет-соединений все, как правило, реализовывалось отдельно в рамках каждой системы и работало по-разному.

Но время шло — и спустя годы приложение стало похоже на большой ком грязи. По ряду уважительных причин (которые я не буду здесь упоминать) команда разработчиков решила, что необходимо перейти на более современную архитектуру. Эти изменения подразумевали в первую очередь очистку кода. И одним из первых шагов было использование фреймворка. Мы выбрали Spring. На то время у нас был Java EE (сейчас это Jakarta EE), но большинство участников проекта решили, что лучше перейти на Spring, так

как этот фреймворк является более простой альтернативой, которую легче реализовать и проще поддерживать.

Переход был не из простых. Вместе с парой коллег, каждый из которых был экспертом в своей области, а также хорошо разбирался в работе самого приложения, мы потратили на это преобразование много сил.

Но результат был потрясающим! Мы удалили более 40 % строк кода. Именно в этот момент я впервые понял, какое серьезное влияние может оказать использование фреймворка.

ПРИМЕЧАНИЕ

Выбор и использование фреймворка зависит от конструкции и архитектуры приложения. Изучая фреймворк Spring, вы поймете, что имеет смысл исследовать также и эти аспекты. В приложении А вы найдете описание архитектур программного обеспечения и ссылки на отличные ресурсы на тот случай, если захотите углубиться в данную тему.

1.2. ЭКОСИСТЕМА SPRING

Рассмотрим Spring и связанные с ним проекты Spring Boot и Spring Data. В книге вы узнаете о них все и вдобавок получите полезные ссылки. На практике часто используют несколько фреймворков одновременно, каждый из которых помогает ускорить разработку определенной части приложения.

Мы называем Spring фреймворком, но в действительности он гораздо сложнее. Spring — это целая экосистема фреймворков. Как правило, когда разработчики упоминают фреймворк Spring, они имеют в виду часть программного функционала, которая включает в себя следующее.

1. **Spring Core** — фундаментальная часть Spring, в которой реализован его базовый функционал. Одной из этих функций является контекст Spring. Как будет подробно описано в главе 2, контекст Spring — это фундаментальная функциональная возможность, благодаря которой Spring может управлять экземплярами приложения. Также частью функционала Spring Core являются аспекты Spring. С ними Spring может перехватывать определенные в приложении методы и манипулировать ими (мы подробно рассмотрим аспекты в главе 6). Еще один компонент, который вы обнаружите в составе Spring Core, — это Spring Expression Language (SpEL). Он позволяет описывать конфигурации Spring с помощью специального языка. Наверняка для вас все это новые понятия — я не ожидаю, что вам приходилось слышать о них ранее. Но вскоре вы поймете, что в состав Spring Core входят механизмы, позволяющие интегрировать Spring в ваше приложение.

2. **Spring MVC (model-view-controller, «модель — представление — контроллер»).** Эта часть фреймворка Spring позволяет создавать веб-приложения, обрабатывающие HTTP-запросы. Мы будем использовать Spring MVC, начиная с главы 7.
3. **Spring Data Access** — еще одна базовая часть Spring. Она предоставляет основные инструменты для соединения с базами данных SQL, что позволяет реализовать уровень доступа к данным в приложении. Мы будем использовать Spring Data Access, начиная с главы 13.
4. **Spring Testing.** Эта часть фреймворка включает в себя инструменты, позволяющие писать тесты для Spring-приложения. Мы рассмотрим эту тему в главе 15.

Для начала вы можете представить фреймворк Spring в виде планетной системы, в которой Spring Core играет роль центральной звезды и притягивает к себе остальные части фреймворка (рис. 1.3).

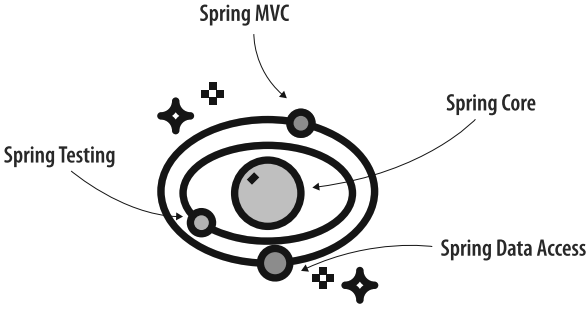

Рис. 1.3. Фреймворк Spring можно представить в виде планетной системы, в центре которой находится Spring Core. Модули программного обеспечения играют роль планет, которые вращаются вокруг «звезды» Spring Core и удерживаются ее гравитационным полем

1.2.1. Spring Core вблизи: ядро Spring

Spring Core — это та часть фреймворка Spring, которая обеспечивает фундаментальные механизмы его интеграции в приложение. Spring работает по принципу *инверсии управления* (inversion of control, IoC): вместо того чтобы приложение само контролировало свое выполнение, управление передается некоторому другому программному обеспечению — в данном случае фреймворку Spring. Посредством системы настроек мы предоставляем фреймворку инструкции о том, как распоряжаться написанным нами кодом, что и определяет логику работы приложения. Именно это и подразумевается под «инверсией» в аббревиатуре IoC: мы не позволяем приложению управлять собственным выполнением

посредством его же кода или использовать зависимости. Вместо этого мы передаем фреймворку (зависимости) управление приложением и его кодом (рис. 1.4).

ПРИМЕЧАНИЕ

В этом контексте «управление» означает такие действия, как «создание экземпляра» или «вызов метода». Фреймворк может создавать объекты классов, определенных в приложении. На основании написанных вами конфигураций Spring способен перехватывать выполнение метода и дополнять его различными функциями, к примеру регистрацией любой ошибки, которая может возникнуть в процессе выполнения этого метода.

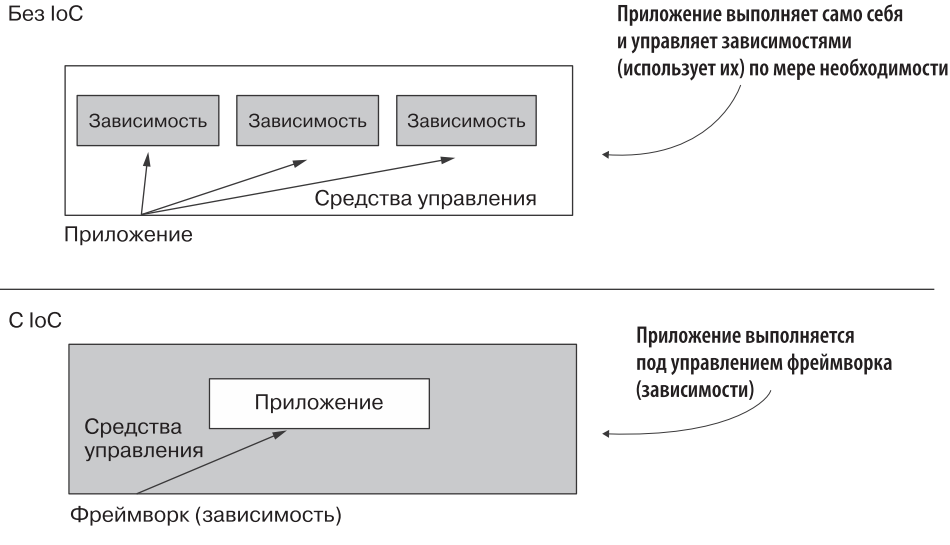

Рис. 1.4. Инверсия управления: вместо того чтобы выполнять собственный код, в котором используется несколько зависимостей, в сценарии IoC приложение выполняется под управлением зависимости. В данном случае выполнением приложения управляет фреймворк Spring. Таким образом, Spring реализует сценарий выполнения IoC

Мы начнем изучение Spring со Spring Core, а изучение Spring Core — с функционала Spring IoC, который рассмотрим в главах 2–5. Контейнер IoC «склеивает» компоненты Spring между собой, а компоненты приложения — с фреймворком. Благодаря контейнеру IoC, который часто называют контекстом Spring, объекты становятся видимыми для Spring, и фреймворк может их использовать в соответствии с заданной вами конфигурацией.

В главе 6 мы продолжим рассматривать аспектно-ориентированное программирование (aspect-oriented programming, AOP) в Spring. Spring позволяет управлять

экземплярами, помещенными в контроллер IoC, и, в частности, перехватывать методы, описывающие поведение этих экземпляров. Такая возможность называется *аспектированием* метода. Spring AOP — один из наиболее типичных способов взаимодействия фреймворка с приложением. Это свойство также делает Spring AOP одной из необходимейших частей фреймворка. В состав Spring Core входят также управление ресурсами, интернационализация (i18n), преобразование типов и SpEL. Все эти функции неоднократно встретятся нам в примерах на протяжении всей книги.

1.2.2. Сохранение данных приложения с помощью Spring Data Access

Для большинства приложений критически важно сохранять часть обрабатываемых данных. Работа с базами данных — фундаментальная часть проекта. В Spring для сохранения данных, как правило, применяется модуль Data Access. В нем используется, в частности, управление транзакциями, JDBC и интеграция с фреймворками, реализующими *объектно-реляционную привязку* (object-relational mapping, ORM), например, с Hibernate (если вы еще не знаете, что такое ORM-фреймворк, или ничего не слышали о Hibernate — ничего страшного; мы рассмотрим эти аспекты далее в книге). В главах 12–14 мы изучим все необходимое, чтобы вы смогли начать работать со Spring Data Access.

1.2.3. Возможности Spring MVC для разработки веб-приложений

Большинство приложений, разрабатываемых на Spring, — это веб-приложения. Поэтому в экосистеме Spring вы обнаружите большой набор инструментов, позволяющих создавать всевозможные веб-приложения и веб-сервисы. С помощью Spring MVC можно писать их в виде обычных сервлетов, что, как правило, и делают для огромного количества современных продуктов. Мы рассмотрим использование Spring MVC более подробно в главе 7.

1.2.4. Тестирование в Spring

Модуль тестирования Spring включает в себя большой набор инструментов, которые мы будем использовать для написания модульных и интеграционных тестов. О тестировании написано немало страниц, но мы в главе 15 рассмотрим лишь то, что нужно для начала тестирования в Spring. Я также дам ссылки на ряд полезных ресурсов, которые стоит изучить, чтобы более детально разобраться в теме. Как показывает мой опыт, нельзя стать настоящим разработчиком, не освоив тестирования, так что вам необходимо в это вникнуть.

1.2.5. Проекты на базе экосистемы Spring

Экосистема Spring — это гораздо больше, чем просто функции, описанные выше. Она включает в себя множество других фреймворков, хорошо интегрированных между собой и образующих целую вселенную. Здесь вы найдете такие проекты, как Spring Data, Spring Security, Spring Cloud, Spring Batch, Spring Boot и т. д. При разработке вы можете использовать сразу несколько из них. Например, можно построить приложение на возможностях Spring Boot, Spring Security и Spring Data. В следующих главах мы будем работать над небольшими заданиями, в которых будем применять функционал различных проектов экосистемы Spring. Под проектом я подразумеваю независимо разработанную часть экосистемы Spring. Каждый из них — плод работы отдельной команды, продолжающей расширять его возможности. Все проекты имеют отдельное описание и свою ссылку на официальном веб-сайте Spring: https://Spring.io/projects.

Но я буду упоминать Spring Data и Spring Boot и за пределами вселенной Spring. Эти проекты часто используются в приложениях, так что важно изучить их сразу.

Расширение возможностей хранения данных с помощью Spring Data

Проект Spring Data — это часть экосистемы Spring, которая позволяет легко подключаться к базам данных и использовать уровень доступа к данным, написав минимальное количество строк кода. Проект обращается как к SQL, так и к NoSQL-базам данных и имеет высокоуровневую обертку, которая упрощает работу с хранением данных.

ПРИМЕЧАНИЕ

В нашем распоряжении есть модуль Spring Data Access, который является частью Spring Core, и Spring Data, который является независимым проектом экосистемы Spring. Spring Data Access обеспечивает реализацию фундаментальных средств доступа к данным, таких как механизм транзакций и инструменты JDBC. Spring Data упрощает доступ к базам данных и предлагает расширенный инструментарий, который делает разработку более понятной и обеспечивает возможность подключения приложения к разнообразным источникам данных. Мы рассмотрим эту тему подробнее в главе 14.

Spring Boot

Проект Spring Boot — это часть экосистемы Spring, реализующая концепцию «соглашения важнее конфигурации». Главная идея этой концепции состоит в следующем. Вместо того чтобы описывать все параметры конфигурации в самом фреймворке, Spring Boot предлагает некую конфигурацию по умолчанию, которую можно изменять по мере необходимости. В результате, как правило, приходится писать меньше кода, поскольку вы следуете известным соглашениям и ваше приложение отличается от других лишь несколькими мелкими деталями.

Поэтому будет более эффективно не описывать заново всю конфигурацию в каждом новом приложении, а начать с некоторой конфигурации по умолчанию и изменить в ней лишь то, что отличается от общепринятого. Мы будем подробно рассматривать Spring Boot, начиная с главы 7.

Экосистема Spring огромна и включает в себя множество проектов. Некоторые из них будут встречаться чаще, чем другие; а некоторые, возможно, вам вообще не понадобятся, если только не придется разработать какое-то специфическое приложение. В этой книге будут упоминаться только те проекты, которые действительно необходимы на начальном этапе: Spring Core, Spring Data и Spring Boot. Полный список проектов экосистемы Spring вы найдете на официальном веб-сайте Spring: https://Spring.io/projects/.

АЛЬТЕРНАТИВЫ SPRING

Если углубиться в серьезное обсуждение альтернатив Spring, то некоторые читатели могут по ошибке посчитать их альтернативами всей экосистеме. Но аналоги есть у многих отдельных компонентов и проектов, входящих в экосистему Spring, — другие фреймворки и библиотеки, коммерческие или с открытым кодом.

Возьмем, к примеру, контейнер Spring IoC. Было время, когда разработчики очень ценили спецификацию Java EE. Обладая немного иной философией, Java EE (код которой в 2017 году был переработан, открыт и назван Jakarta EE — https://jakarta.ee/) предлагает такие спецификации, как Context and Dependency Injection (CDI) и Enterprise Java Beans (EJB). CDI и EJB можно использовать для управления контекстом экземпляров объектов и для реализации аспектов (которые в терминологии EE называются interceptors — «перехватчики»). Долгое время хорошим фреймворком для управления экземплярами объектов в контейнере был Google Guice (https://github.com/google/guice).

Для некоторых проектов удается найти одну или несколько альтернатив. Например, вместо Spring Security можно использовать Apache Shiro (https://shiro.Apache.org/). Или же вместо Spring MVC и технологий Spring можно построить веб-приложение на фреймворке Play (https://www.playframework.com/).

Из более новых проектов многообещающе выглядит Red Hat Quarkus (https://quarkus.io/). Quarkus был создан для внедрения облачных решений и сейчас быстро совершенствуется. Я не удивлюсь, если однажды он станет одним из ведущих проектов для разработки промышленных приложений в экосистеме Java.

Мой вам совет: всегда рассматривайте альтернативы. При разработке программного обеспечения необходимо держать глаза открытыми и никогда не верить в одно решение, «единственное и неповторимое». Вам постоянно будут встречаться сценарии, в которых одна технология работает лучше, чем другая.