ИССЛЕДОВАНИЕ МЕТОДИКИ ИЗУЧЕНИЯ ПРОГРАММНОГО ФРЕЙМВОРКА С ПОМОЩЬЮ АКТИВНЫХ ТУТОРИАЛОВ

Баранов Я.В. Email: Baranov1146@scientifictext.ru

Баранов Ярослав Викторович – аспирант,
кафедра информатики и прикладной математики, факультет программной инженерии и компьютерной техники,
Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики, г. Санкт-Петербург

Аннотация: *в статье подробно разобраны существующие подходы к изучению программных фреймворков на примере Spring Framework. На основе этих подходов выработана новая методика обучения, которая является максимально (95%) нацеленной на практику. Также в рамках данной статьи представлена модель языка разметки KML, который существенно ускоряет и автоматизирует создание активных туториалов. В дополнение к этому в статье представлено 5 учебных задач, которые покрывают Spring Jdbc/Transactions на 50% от необходимых для сертификации знаний.*
Ключевые слова: *фреймворк, активный туториал, эффективное обучение, KML.*

LEARNING TECHNIQUES BASED ON ACTIVE TUTORIALS

Baranov Ya.V.

Baranov Yaroslav Viktorovich – post graduate Student,
DEPARTMENT OF INFORMATICS AND APPLIED MATHEMATICS, FACULTY OF SOFTWARE ENGINEERING AND COMPUTER TECHNOLOGY,
SAINT PETERSBURG NATIONAL RESEARCH UNIVERSITY OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS, SAINT PETERSBURG

Abstract: *the article observers existing approaches for learning software frameworks. It uses Spring Framework as a reference. Based on that, we developed a new learning technique which is practice-oriented (95%). In addition, we developed a new markup language (KML), and the article displays a few examples of KML code. It accelerates and automates the creation of active tutorials. Also, the article presents five Spring Jdbc/Transactions tasks. They cover half of the knowledge necessary for obtaining a Spring certificate.*
Keywords: *framework, active tutorial, efficient learning, KML.*

УДК 331.225.3

Введение

Spring Framework традиционно считается одним из самых сложных и при этом высокооплачиваемых. Однако, существующие подходы к его изучению крайне неэффективны. Мы хотели бы предложить вам принципиально новый подход к изучению спринга, который заключается в использовании концепции активных туториалов. Они представляют собой сбалансированный набор инструкций, которые поддерживают внимание разработчика и проводят его через те или иные аспекты фреймворка.

В рамках статьи представлено 5 учебных задачек, которые покрывают Spring Jdbc/Transactions на 50% от необходимых для сертификации знаний.

Традиционные подходы к обучению Spring

В изучении фреймворков есть 2 крайности. Первая — когда человек работает в компании и просто делает те задачи, которые даёт заказчик. Это медленный, очень медленный способ развития, но по нему идут большинство разработчиков. Логично делать только то, что просят, особенно когда за это платят, не так ли? Однако, обучение «на работе» лишь кажется эффективным. Большинство задач, которые решают современные программисты, заключается в сопровождении систем. Всё уже написано до нас, нужно лишь фиксить баги\править конфиги, и дорабатывать систему напильником. В этом нет ничего плохого, но вот обучение самим технологиям происходит крайне медленно. Конечно, рано или поздно вам придётся раскопать документацию спринга [1] и вы запомните всё необходимое, но на это уйдут годы. Может быть, стоит попробовать сначала «накрутить» знания и опыт, а потом уже брать практические задачки посложнее (и за существенно больший оклад, разумеется)? (С Java это точно работает, можно за 2-3 месяца целенаправленно изучить Java SE и это могут засчитывать за год-другой опыта. Много знакомых с универа, кто так делал).

Вторая крайность — это целенаправленное обучение. Это быстрый, но очень тяжёлый способ. Он заключается в том, что человек продвигается по книге (или курсу), а потом пытается применить это на практике и как-то запомнить. И если с Java SE [2] такой подход ещё кое-как работает, то со спрингом всё глухо и туго. Даже в самых лучших книгах зачастую не объясняется, где конкретно применять те или иные особенности, приходится догонять это самому. Но самое обидное тут — это забывание

информации, добытой таким тяжёлым трудом. Одна из проблем обучения — это отсутствие эффективного повторения. Если вы изучали спринг классическим способом (например, читали книгу и пробовали код на практике), то на это были потрачены огромные усилия, но БЕЗ возможности восстановления. Простое перечитывание книги через 1-2 года не вернёт вам забытой информации (которую вы получили через практику, параллельную с прочтением). Возникает некая дилемма — как же сделать так, чтобы было много практики, но при этом программист «направлялся» в нужные области?

Проблемы обучения

Итак, мы пришли к выводу, что крайности в обучении — это плохо. Давайте подумаем, как можно оптимизировать процесс. Целенаправленное обучение выглядит правильной идеей, которую нужно развивать дальше. Рассмотрим более подробно, почему же это вызывает трудности у большинства людей.

Для начала, выберем материалы, по которым стоит обучаться. Главное требование к ним — ограниченный набор информации (только то, что нужно для сертификации/собеседования/практики), а также последовательная и логически взаимосвязанная подача этой информации. По данным критериям, наилучшим образом подходят книги (Спринг в действии, Учебное пособие по сертификации от Юлианы и т.д.) Они очень хорошо продуманы и оформлены (на мой взгляд, куда приятнее и подробнее, чем видеокурсы по спрингу на udemy). Казалось бы, читай себе, вникай, пробуй, экспериментируй — и будут знания! Но не тут-то было.

Дело в том, что сам процесс чтения книги и разбора теории в ней очень плохо состыкован с процессом апробации этой теории на практике. Он не естественный. Какой бы ни была книга идеальной, она остаётся книгой. Она по своей природе не предназначена для обучения программированию. Программист, в конечном итоге, должен набивать хорошие привычки по сознательному использованию тех или иных технологий фреймворка. Между «Я прочитал и понял» и «Я умею это применять и знаю где» образуется огромная пропасть. Чтобы преодолеть её, приходится вложить немалые усилия. Скажу честно — я написал довольно много веб-приложений на Spring, однако всё равно испытывал множество трудностей при прочтении глав книги «Спринг в действии». На данный момент я детально разобрал около 30% из обеих книг, и готов к сертификации Spring 5 примерно на 60%.

Процесс связывания представленной в книге информации с привычками — очень тяжёлый и не эффективный. Он часто вызывает множество негативных эмоций (когда что-то не работает или не запускается), заставляет много гуглить и решать проблемы. Неужели нельзя сделать некоторые «пути (trails)», по которым можно будет провести разработчика, показать ему правильные решения, да так, чтобы он сам их делал? Ведь опытные разработчики прекрасно знают о таких путях. Для них это что-то очевидное. Но тут возникает проблема в форме передачи знаний. Как мы уже выяснили, книги не подходят для этой цели.

Ещё одна проблема, на которую хотелось бы обратить внимание — это проблема повторения. Даже если мы преодолеем все трудности и широко прокачаем знания спринга, со временем они будут угасать (не считая того небольшого процента, который получилось связать с текущими задачами). Человеку свойственно забывать, и с этим ничего не поделать. Единственное, что мы можем сделать — это попробовать оптимизировать повторения. Когда я учился в бакалавриате, я возлагал большие надежды на теорию интервального повторения [3] и хранение знаний в виде флеш-карт. Однако, флеш-карты слишком независимы друг от друга (не подходят для хранения связанных знаний о спринге). Даже для изучения Java их эффективность средняя. Да и они тоже не естественны, ведь программист должен повторять через практику.

Я многие годы ломал голову над тем, как сделать обучение 100% завязанным на практику (как следствие — не скучное и с высоким КПД). Сейчас я коротко расскажу о вариантах, которые были перепробованы. Для начала, я пытался найти набор подробных и учебных ТЗ, которые бы, кроме задач давали бы технические наводки (разработать такой-то REST Api используя такие-то классы). Ничего дельного я не нашёл, и потратил кучу времени. Да, я видел отдельные авторские курсы по некоторым частям спринга, но полного покрытия сертификации нигде не собрать. Да и «книжная» проблема этих туториалов остаётся (хоть и некоторые авторы пытались сделать пошаговые руководства, они всё равно имеют недостатки).

Одна из проблем книг и туториалов — их очень скучно и нудно читать. Это ещё больнее, чем гуглить баги. Я не хочу читать, я хочу кодить! Что, если использовать книгу только для наводки на темы и названия классов, а потом уже самостоятельно (через эксперименты и гугл) догонять всё остальное? А уже потом и главу перечитывать, пересиливая себя и собирая оставшиеся крупицы знаний. Собственно, я так и изучал спринг. Не с начала главы (унылое введение), а с середины, пытаясь за что-то ухватиться и экспериментировать вокруг этого. IDE очень помогает в этом с помощью автозаполнения, просмотра JavaDoc и исходников, удобной отладки. Я бы назвал это «изучение с помощью экспериментов с API». Я даже развил ряд особых методик вокруг этого метода, но там всё равно остаются некоторые фундаментальные проблемы.

А именно, проблема «ступоров» никуда не девается. Она вызывает по-прежнему много боли, хоть это уже более естественно и приближенно к практике (в реальных проектах придётся много таких ступоров решать, прокачать навык будет полезно). На самом деле, на этой методике можно вполне себе дойти до конечной цели (сертификация). Но это будет требовать очень много усилий, в 3-4 раза больших, чем если бы идти по накатанной дорожке. Да и проблема повторения информации всё равно остаётся. И хочется сделать что-то более приближенное к идеальному.

Давайте подумаем, как происходит обучение по классическому (большей частью пассивному) туториалу. Большинство туториалов по Spring просто отвратительны по своей структуре (включая гайды на spring.io). Самый большой их недостаток, который я просто терпеть не могу — это линейность. Многие авторы «вываливают» большие куски кода, которые нужно копипастить к себе. Было бы правильнее начинать с простого примера (минимально возможной демонстрации, которую можно запустить и поэкспериментировать), и потом накручивать на него разные навороты. Принцип «от простого к сложному» — золотой закон обучения! Но нет ведь. Каждый автор считает нужным накрутить информации в 2-4 раза больше чем нужно, по кускам это разбирать и только потом запускать.

К примеру, откроем руководство по поднятию SOAP-сервиса на спринге [4]. Они тут и spring-boot добавили, и wsdl4j с процессом генерации Java-классов с использованием gradle, и целый in-memory репозиторий CountryRepository (хотя простой строки «Hello world» мне хватило бы). И только в самом конце объяснили, как с помощью curl запустить всю эту систему. Нет, я конечно всё понимаю — авторы хотели дать наглядный пример «всё в одном» и разобрать его. Но, с точки зрения понимания информации, такой подход не годится.

Идея активных туториалов

Что, если сделать некое подобие виртуального учителя, который просто даёт маленькие указания типа "сделай то и это", а в случае, если программист ошибся или забыл — учитель просто даёт фрагмент кода. По своей сути, задачи в рамках данного проекта и являются таковыми. Суть их в том, что у нас есть маленький набор инструкций, к каждой из которых есть ответ и он под спойлером (скрыт). Разницу между ними и обычным туториалом можно увидеть рисунке 1.

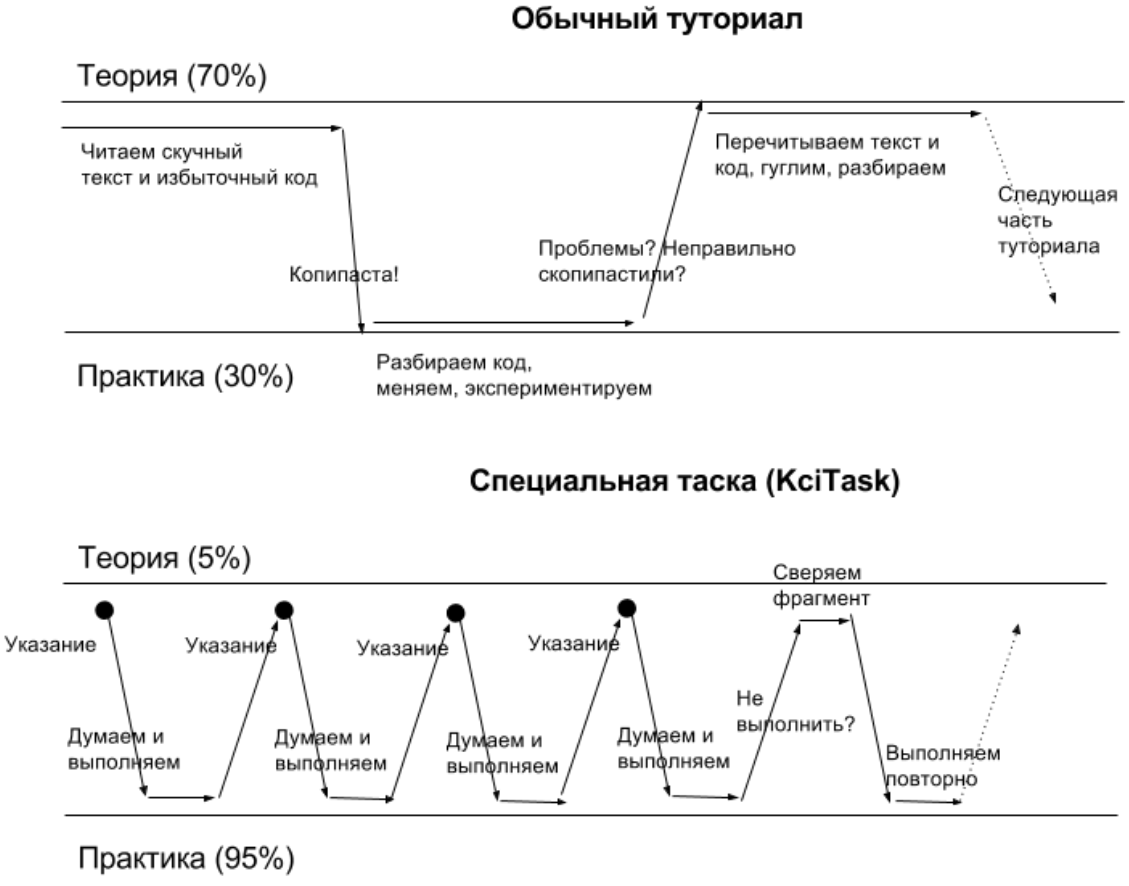

Рис. 1. Сравнение классического туториала и ориентированного на практику

Концепция проекта с чистого листа

Перед тем, как продолжить разбор активных туториалов, хотелось бы рассказать об одной важной концепции, на которой они базируются. Эту концепцию вы можете использовать совместно с любой

методикой обучения, но почему-то о ней редко пишут или упоминают где-то. Так вот, суть её в том, что тренировочные проекты нужно делать с нуля. Никаких start.spring.io, каждый день заходите в File->new Project->Hello world и на нём базируете ВСЕ ваши проекты, включая веб. И все maven-зависимости тоже забиваете по-новому. Благодаря этому вы запомните зависимости между спринг-модулями, зачем нужен каждый из них и т.д. На практике это очень сильно пригождается, когда есть какие-то проблемы с зависимостями.

Любите ли вы XML? Авторы обеих книг по Spring соглашаются, что рано или поздно XML станет пережитком прошлого. Однако, они сами приводят большинство решений в двух вариантах (XML+Аннотации). Я не любил XML до тех пор, пока не устроился в большую компанию. Сейчас это просто часть работы. Слишком много готовых решений сделано, которые просто пронизаны XML и переписать их без него — потратить огромные деньги и получить несравнимо мало. Никто не будет этого делать. Поэтому, я старался чередовать XML/Аннотации в моих задачках, что и вам рекомендую. Если правильно обучаться (по описанной в статье методике), то XML не вызывает проблем, а, напротив, помогает взглянуть с другой стороны на некоторые решения и лучше их запомнить. Написание XML кода (с автозаполнениями и подсказками, с помощью IDE) также приятно, как и написание Java кода.

Каждая инструкция в активном туториале должна быть выполнимой БЕЗ копипасты. Современные IDE позволяют умножить её на ноль. Да, даже beans.xml со всеми его приблудами, даже dependencies — всё можно сделать внутри IDE. Это намного приятнее, чем бездумно копировать код. Как я уже сказал, я хочу сделать обучение приятным и это одно из проявлений.

Каждая инструкция в кси-таске заставляет вас немножко подумать и что-то вспомнить. В этом и заключается «активность» такого туториала. Этот процесс намного приятнее, чем чтение или копирование кода. Тут нужно поддерживать баланс — инструкция не должна быть слишком тупой (иначе это будет не так приятно), и не должна быть слишком сложной (что повлечет за собой большие куски кода и проблемы, аналогичные с туториалами). Я нигде не видел подобных разработок, хоть и повидал много разных систем обучения.

Специальный язык для активных туториалов – KML

Одна из причин, по которым активные туториалы до сих пор никто не распространил — отсутствие формата для их хранения. Существующие языки разметки совершенно не подходят для перемешивания кода и текста. Первую версию KciTasks я сделал как надстройку на HTML, и это было просто ужас как неудобно! Потом я сделал свой небольшой язык разметки, который отличным образом подходит для тасок и компилируется в HTML. И происходит это прямо во время загрузки веб-страницы. Вот примеры:

Пример — Создание бина JdbcTemplate

```
=Create a @@JdbcTemplate@@ as a @@@Bean@@ with @@DataSource@@ injected into it
+Main.java
@Bean
JdbcTemplate getJdbcTemplate(DataSource dataSource){
   return new JdbcTemplate(dataSource);
}
```

Фрагмент кода 1

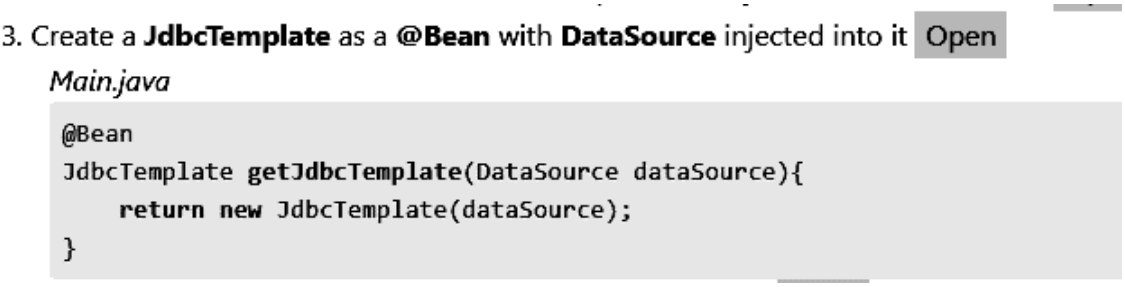

Рис. 2. Визуализация фрагмента кода 1

Заключение

Я вижу два пути, по которым можно использовать KciTasks. Первый — использовать технологию для углублённого изучения Spring Framework. На мой взгляд, это должно быть эффективно. Вы просто выполняете инструкцию за инструкцией, сверяясь с решением и корректируя себя. Поначалу вы будете много «подглядывать» туда, но это нормально. На следующий день попробуйте подглядывать как можно меньше и всё делать самому. Все таски рассчитаны на то, чтобы быть сделанными без разворачивания спойлеров. Таски самодостаточны, достаточно всего лишь вчитаться в инструкцию и решение к ней, и вывести знания из экспериментов.

Второй путь — использовать KciTasks для повторения. Когда вы изучаете какой-либо фрейм или язык программирования по книге/курсу, вы вкладываете много усилий. И пусть они не пропадут даром

— вложите весь полученный опыт в таски, чтобы потом через полгода можно было пройти по проторенной дорожке и всё вспомнить.

Список литературы / References

1. Spring Framework reference // Spring Framework. [Электронный ресурс]. Режим доступа: https://docs.spring.io/spring/docs/current/ (дата обращения: 22.04.2018).
2. Java SE Downloads // Oracle. [Электронный ресурс]. Режим доступа: http://www.oracle.com/technetwork/java/javase/downloads/index.html/ (дата обращения: 22 апреля 2018).
3. Friendly, intelligent flash cards. Remembering things just became much easier // Anki. [Электронный ресурс]. Режим доступа: http://ankisrs.net (дата обращения: 22.04.2018).
4. Producing a SOAP web service // Spring Framework. [Электронный ресурс]. Режим доступа: https://spring.io/guides/gs/producing-web-service/ (дата обращения: 22.04.2018).