Санкт-Петербургский государственный университет

Программная инженерия
Кафедра системного программирования

Осипова Александра Вадимовна

Разработка библиотеки для динамической конфигурации промышленных Java-приложений

Курсовая работа

Научный руководитель:
ст. преп. Я. А. Кириленко

Технический консультант:
ст. программист DSXT
А. М. Плотников

Санкт-Петербург
2020

Оглавление

Введение

Окружающая среда адаптируется к изменениям, вносимым человеком. Программа меняет своё поведения с изменением своих параметров. В первом случае нет необходимости останавливать течение времени, чтобы изменения вступили в силу. Поэтому естественно желание, меняя параметры программы, видеть изменения в приложении без его перезапуска, особенно если к приложению предъявляются особые требования в отношении его доступности.

В то же время с каждым годом сложность разработки программных продуктов возрастает, возрастает и количество сторонних сущностей, с которыми приложение взаимодействует (базы данных, HTTP Services и т.д.), в частности существует тренд на использование микросервисов, и большая часть архитектурных проблем перекладывается на инфраструктуру, связь с которой требует конфигурации.

Поэтому, когда речь заходит о крупных проектах, неудивительна необходимость конфигурировать большое количество параметров (различные ссылки, бизнес-логика и прочее). Часть таких параметров подвергается изменениям редко, но часть может потребоваться изменять часто (например, при А/Б тестировании).

Цель данной курсовой работы — разработка библиотеки для динамической конфигурации. Работа возникла из желания компании DSX Technologies иметь более простой и функциональный инструмент конфигурации промышленных Java-приложений.

1. Постановка задачи

Целью данной работы является создание библиотеки для динамической конфигурации промышленных Java-приложений. Для её достижения были поставлены следующие задачи:

- Провести исследование предметной области и сделать обзор существующих решений
- Спроектировать и реализовать механизмы для динамической конфигурации Java-приложений
- Поддержать возможность использования различных источников конфигурации (файловая система, база данных и т.п.)

2. Обзор существующих решений

В мире Java существует множество библиотек для конфигурации приложений. Основными факторами для включения в данный обзор являются наличие поддержки перезагрузки конфигурации во время работы приложения и отсутствие сторонней специфичной функциональности (например, инъекция зависимостей и поддержка IoC в Spring[1]).

1. Cfg4j [1]. Одно из последних решений в данной области. Поддерживает Consul, Git repos (*.yaml, *.properties), различные файлы. Библиотека, в которой функциональность динамической перезагрузки планировалась изначально, но ввиду особенностей архитектуры (не изменения распространяются в конфигурируемые объекты, а те обращаются в конфигурационное хранилище) появляется проблема с конфигурированием параметров, от которых зависят нетривиальные объекты или ресурсоёмкие вычисления [10].

2. Apache Commons Configuration [2]. Библиотека поддерживает множество форматов файлов (*.ini, *.properties, *.xml ...) и JDBC Datasource, обеспечивает типизированный доступ к параметрам (с одним или несколькими значениями), но возможность перезагрузки во время работы программы добавлялась не сразу, поэтому механизм получился громоздкий: три интерфейса, реализованные только для файлов, довольно сложный контракт их взаимодействия [3, 10]. Проблема с конфигурированием нетривиальных сущностей также присутствует.

3. ReactiveConfig [10]. Осенью 2018 года на конференции Joker был представлен доклад о библиотеке конфигурации ReactiveConfig, написанной на Scala для компании Tinkoff. В ней присутствует динамическая перезагрузка, валидация значений, также архитектура хороша для перезагрузки ресурсоёмких вычислений и сложных сущностей, но данной библиотеки нет в открытом доступе и,

[1] https://spring.io

так как она разрабатывалась под проекты на Scala, использовать её при работе с другими языками JVM может быть затруднительно.

	Java	Open-source	Перезагрузка сложных сущностей
Apache Commons Configuration	+	+	-
Cfg4j	+	+	-
ReactiveConfig by Tinkoff	+/-	-	+

Рис. 1: Сравнительная таблица существующих решений

Таким образом, ни одно из решений не подходит для удобной конфигурации промышленных Java-приложений во время их работы, в частности, в Cfg4j и Apache Commons Configuration затруднено конфигурирование сложных сущностей и параметров, изменение которых влечет ресурсоёмкие вычисления, а библиотека ReactiveConfig, помимо того, что отсутствует в открытом доступе, не приспособлена для приложений, написанных на Java.

Поэтому и было принято решение отказаться от интеграции одного из существующих решений и разработать новую библиотеку для динамической конфигурации в рамках данной курсовой работы.

3. Описание реализации

В данном разделе описывается реализация библиотеки и рассказывается о концепциях, лежащих в её основе. Исходный код проекта опубликован на хостинге GitHub[2].

3.1. Реактиное программирование

Во время работы приложения невозможно определить наперёд, когда произойдут изменения конфигурационных параметров. Также объекты, зависящие от конфигурации, в процессе работы должны реагировать на эти изменения. Поэтому события изменения конфигурационных параметров и обработки этих изменений *асинхронны*, а изменения конфигурации естественным образом представляются в виде асинхронного потока данных[3].

Таким образом, для задачи обработки изменений логично рассматривать подходы, работающие с асинхронными потоками данных и способные доставлять изменения до зависимых объектов. Именно таким подходом являются реактивное программирование[4] [11, 9] и использование имеющихся реализаций [5, 6] стандарта Reactive Streams [8].

Благодаря реактивному программированию, при котором изменения конфигурации — это реактивный поток, распространяющийся в зависимые от конфигурации объекты, появляется возможность сделать библиотеку конфигурации с акцентом на перезагрузке (например, должна присутствовать адекватная перезагрузка конфигурации нетривиальных сущностей), что явлется особенностью в сравнении с ныне доступными решениями.

[2]https://github.com/dsx-tech/rhea

[3]Под асинхронным потокам данных понимается поток, в который значения попадают одно за другим с произвольной временной задержкой между значениями

[4]Реактивное программирование — парадигма программирования, ориентированная на потоки данных и распространение изменений

3.2. Выбор языка для реализации

Так как речь идёт о конфигурации Java-приложений, выбор основного языка программирования был между языками Java и Kotlin. Главный критерий для сравнения — реактивные возможности. Рассматривались стандартные реализации реактивных потоков в обоих язык [5, 6], но принимая во внимание желание использовать библиотеку в том числе и для приложений, написанных на 8 версии Java, реализация стандарта Reactive Streams в Java оказалась неподходящей, потому что появилась только в 9 версии. Поэтому оставались варианты использования либо сторонней библиотеки RxJava[5], либо Kotlin Flow [5]. Вместе с Дмитрием Вологиным, студентом СПбГУ 2 курса, были опробованы в работе оба варианта, и принято решение использовать Kotlin Flow: данная реализация примитивнее и в то же время её реактивной функциональности достаточно для решения поставленной задачи, к тому же Kotlin очень лаконичный язык и позволяет строить удобные DSL.

3.3. Архитектура

На Рис. 2 представлена диаграмма, описывающая архитектуру библиотеки. Основа архитектуры — асинхронные потоки данных: изменения, асинхронные события, из источников конфигурации распространяются в конфигурируемые объекты. Проект разбит на модули: ядро и реализации интерфейса ConfigSource для некоторых источников. Модули, отвечающие за интеграцию источников, подключаются опционально, поэтому нет избыточных зависимостей.

Главные элементы ядра: интерфейс ConfigSource, классы Reloadable и ReactiveConfig. Созданный объект ReactiveConfig отвечает за некоторые конфигурационные параметры приложения. В нем содержится канал, принимающий изменения параметров от подписанных на него источников конфигурации. Чтобы зарегистрировать параметр необходимо указать его ключ и тип — объект PropertyType. PropertyType содержит в себе значение по умолчанию и функцию, с помощью которой

[5]https://github.com/ReactiveX/RxJava

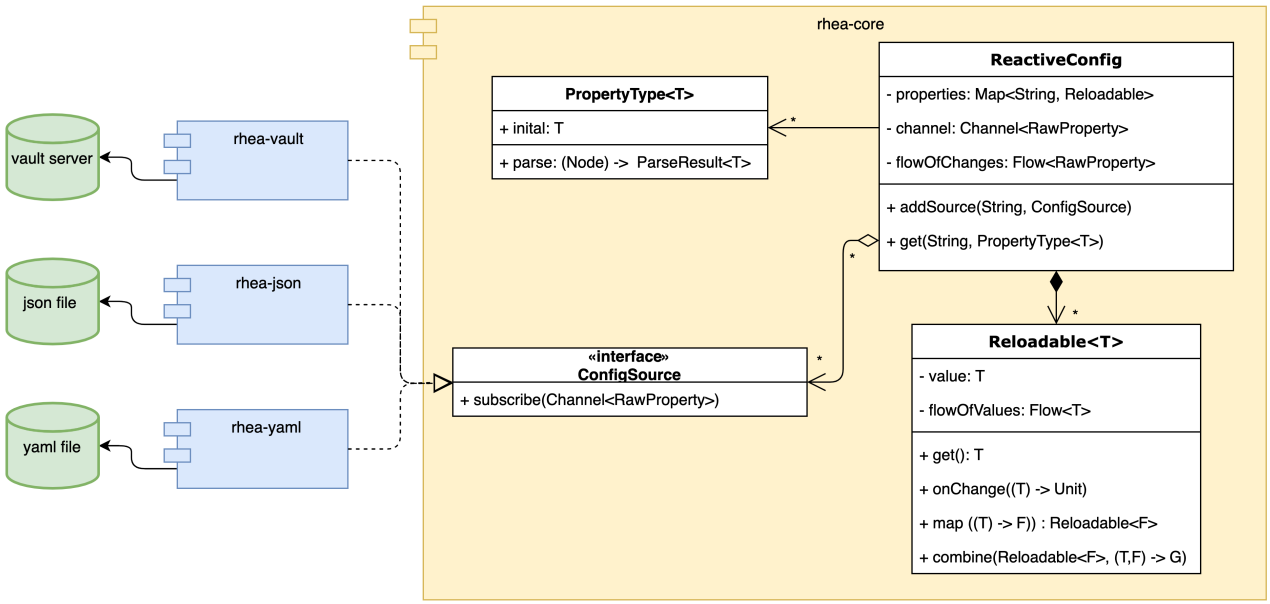

Рис. 2: UML диаграмма основных классов и модулей разработанной библиотеки

происходит разбор значения, пришедшего из источника. Реализованы стандартные типы параметров (stringType, intType и др.), также можно добавлять свои типы, имеющие более сложную структуру. Выглядит объявление перезагружаемого параметра так:

```
Reloadable<Integer> port = config.get(''port'', PropertyTypesKt.intType);
```

При создании начальное значение параметра запрашивается у источника, что позволяет использовать созданный Reloadable без ожидания его полной инициализации. Reloadable содержит в себе реактивный поток, в котором находятся изменения конкретного параметра, за счет чего появляется возможность при обращении выдавать самое «свежее» значение параметра, «пробрасывать» изменения дальше, в созависимые сущности, а также реализовывать нетривиальную логику обработки изменений.

3.4. Источники конфигурации

С помощью реализации интерфейса ConfigSource можно добавлять свои источники конфигурации.

В рамках данной работы в качестве источников конфигурируемых параметров были добавлены файлы типа *.properties, *.yaml, *.json. Для отслеживания изменений используется WatchService API [7], события которого определяют, когда перечитывать файл, обновлять значения и пересылать их дальше, в конфигурируемые объекты.

Также, помимо поддержки конфигурации из файлов, реализована интеграция сервиса Vault [4]. В промышленных приложениях, чтобы минимизировать риски безопасности, возникает потребность хранить секреты (это могут быть пароли, ключи шифрования, сертификаты и т.д). Vault занимается защитой такой информации и контролем доступа к ней. Секреты хранятся в виде ключ-значение. Доступ к хранилищу осуществляется исключительно через API. В качестве непосредственного хранилища зашифрованных данных можно выбрать один из уже поддерживаемых вариантов (например, Consul, MySQL, S3), либо интегрировать что-то своё.

4. Заключение

В ходе выполнения работы получены следующие результаты:

- Сделан обзор предметной области и существующих решений
- Реализована библиотека динамической конфигурации
- Поддержана возможность использования различных источников конфигурации

Список литературы

[1] Cfg4j. Cfg4j. — URL: http://www.cfg4j.org/ (online; accessed: 08.12.2019).

[2] Foundation The Apache Software. Commons Configuration. — URL: https://commons.apache.org/proper/commons-configuration/ (online; accessed: 08.12.2019).

[3] Foundation The Apache Software. Commons Configuration. — URL: https://commons.apache.org/proper/commons-configuration/userguide/howto_reloading.html (online; accessed: 18.12.2019).

[4] HashiCorp. Vault. — URL: https://www.vaultproject.io (online; accessed: 15.05.2020).

[5] Kotlin. Flow. — URL: https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-flow (online; accessed: 10.12.2019).

[6] Oracle. Flow // Java™ Platform Standard Ed. 9. — URL: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html (online; accessed: 10.12.2019).

[7] Oracle. WatchService. — URL: https://docs.oracle.com/javase/tutorial/essential/io/index.html (online; accessed: 16.05.2020).

[8] Reactive Streams. — URL: https://www.reactive-streams.org/ (online; accessed: 10.12.2019).

[9] Staltz Andre. Reactive programming // Github. — URL: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (online; accessed: 10.12.2019).

[10] Tinkoff. ReactiveConfig // Joker. — 2018. — URL: https://www.youtube.com/watch?v=uU1DHBEFVP4&list=PLVe-2wcL84b_DvLWtURPD0Dz2NZil55XI&index=40 (дата обращения: 08.12.2019).

[11] Wikipedia. Reactive programming // Wikipedia. — URL: https://en.wikipedia.org/wiki/Reactive_programming (online; accessed: 10.12.2019).