

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21Б.10-мм

Расширение системы сборки Quarkus проектов для OpenJDK CРaC

Хажоян Левон Владимирович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования СПбГУ Козлов А.П.

Санкт-Петербург
2022

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. OpenJDK CRaC	6
2.2. Инструменты сборки фреймворка Micronaut для CRaC и сравнение с текущей реализацией Quarkus с CRaC	7
2.2.1. Сборка Quarkus приложения для OpenJDK CRaC	7
2.2.2. Сборка Micronaut приложения для OpenJDK CRaC	8
Заключение	10
Список литературы	11

Введение

Java входит в четвёрку самых популярных языков программирования, согласно индексу TIOBE [17].

Для Java существует множество реализаций спецификации платформы, которые называются JDK (Java Development Kit) [18], которая представляет из себя комплект разработчика, включающий компилятор Java (javac), стандартные библиотеки Java классов, различные утилиты и исполнительную систему Java (JRE). OpenJDK — проект по созданию полностью совместимого JDK, состоящего из свободного и открытого исходного кода [9].

Java часто используется в таких индустриях, как мобильная разработка, финансовые системы, анализ данных и встраиваемые системы — там, где от программ требуется производительность. С производительностью у Java-программ есть следующие проблемы: во-первых, долгий старт [20], во-вторых, необходимость предварительного разогрева приложения [11]. Одна из причин популярности языка Java — это JVM (Java Virtual Machine). Она предоставляет управляемый жизненный цикл Java программ, взяв на себя ответственность за исполнение многих обязанностей разработчика. К примеру, управление и оптимизация памяти, ускоряя при этом производительность программ.

Первая проблема связана с тем, что каждая программа состоит из множества классов, которые необходимо загрузить, проинициализировать, определить семантику байткодов, затем исполнить их. Одна из оптимизаций для решения проблемы долгого старта реализована JIT-компиляция [5] (Just-in-Time-компиляция), которая собирает статистику о вызове методов во время исполнения программы. Часто используемые методы компилируются вскоре после запуска виртуальной машины Java, а менее используемые методы скомпилируются намного позже или вообще не выполняются, что даёт прирост производительности, однако тоже требует ресурсы на выполнение.

Другое решение проблемы — технология AppCDS [7] (Application Class Data Sharing), которая помогает ускорять запуск и экономить па-

мать за счет архивации загруженных классов. Однако использование этой технологии даёт лишь небольшой выигрыш [10].

Вторая проблема частично решается Ahead-of-Time [1] компиляцией (то есть компиляции перед исполнением), которая позволяет предсгенерировать нативный код, который можно использовать вместо траты времени на JIT-компиляцию. Однако этот процесс тоже требует времени.

Существует множество технологий, упрощающих и ускоряющих жизненный цикл Java программ [15], одной из таких является фреймворк Quarkus. У приложений на Quarkus довольно высокая скорость запуска, а также низкое потребление памяти [16]. Фреймворк предоставляет множество полезных функций для разработчиков, например, горячая перезагрузка (позволяет вносить изменения в код приложения во время его выполнения без необходимости приостанавливать его вручную или создавать точку остановки) [19], AOT компиляцию при помощи GraalVM Native Image [4, 2], AppCDS [12] и многое другое [14]. Также Quarkus предоставляет удобную работу с проектом, ускоряющим старт уже разогретого Java приложения — CRaC. В данный момент существуют эксперименты показывающие, что приложения стартуют намного быстрее с OpenJDK CRaC [8], чем без него. Однако, проекты Quarkus для OpenJDK CRaC не обладают универсальными шагами сборки и оказывается довольно неудобны в использовании.

1. Постановка задачи

Целью данной учебной практики является расширение системы сборки Quarkus [15] проектов для поддержки жизненного цикла приложений при использовании проекта OpenJDK CRaC.

Для достижения цели были поставлены следующие задачи:

1. сделать обзор проекта OpenJDK CRaC;
2. произвести обзор инструментов сборки фреймворков Micronaut и Quarkus, имеющих поддержку CRaC.
3. сформулировать требования к создаваемому шагу сборки;
4. реализовать шаг сборки;
5. произвести апробацию решения;

2. Обзор

В этом разделе представлен обзор существующих инструментов сборки фреймворка Micronaut и проекта OpenJDK CRaC.

2.1. OpenJDK CraC

Проблемы медленной работы и старта Java приложений:

1. Долгая инициализация приложения:
 - загрузка классов
 - инициализация классов
 - JIT-компиляция
2. Обработка запросов — имеет все те же проблемы, поэтому требует разогрева на каких-то данных.

Coordinated Restore at Checkpoint

В качестве одного из решений проблем медленного старта и работы приложений на Java на основе OpenJDK был создан проект CRaC (Coordinated Restore at Checkpoint) [8].

CRaC предоставляет две операции:

- checkpoint — сохранение состояния процесса в образ
- restore — восстановление оригинального процесса

Однако в реальности, Java приложения часто открывают внешние соединения, которые в открытом состоянии сохранить в образ невозможно. Поэтому создание checkpoint-а может повлечь за собой разорванные открытые соединения, что для работающих приложений может стать критично. Информация как открыть только что разорванные соединения может утечь, а значит, checkpoint будет прерван.

Для этого в CRaC API существует метод разрешения опасных состояний, а именно, `ResourceManager`. `ResourceManager` реализует два

метода, способные управлять внешними ресурсами, для достижения безопасного сохранения образа:

- `beforeCheckpoint` — с помощью него пользователь может описать, какие соединения он оборвёт перед checkpoint-ом
- `afterCheckpoint` — с помощью него пользователь может описать, какие внешние ресурсы он восстановит после restore-a

Использование этих операций на оптимальной производительности предоставляет запуск уже разогретой виртуальной машины, за счет того, что операция `restore` развернет оригинальный процесс из сохраненного образа. При этом сам образ при создании, включал в себя загруженные и проинициализированные классами, готовые JIT-компиляции, потоки и их состояния и кучу.

Таким образом, CRaC решает ранее обозначенные проблемы Java-программ. Если выполнялась операция `checkpoint`, CRaC обязуется восстановить процесс в первоначальном виде, что говорит о надежности сохраненных образов.

2.2. Инструменты сборки фреймворка Micronaut для CRaC и сравнение с текущей реализацией Quarkus с CRaC

2.2.1. Сборка Quarkus приложения для OpenJDK CRaC

В данный момент сборка приложения [13] Quarkus для OpenJDK CRaC выполняется вручную через представленные шаги:

Запускаем JDK в `checkpoint` режиме:

```
$JAVA_HOME/bin/java -XX:CRaCCheckpointTo=cr  
-jar target/example-quarkus-SNAPSHOT.jar
```

Разогреваем JVM, путём вызова ста тысяч элементарных запросов :

```
siege -c 1 -r 100000 -b http://localhost:8080/hello
```

Создаём checkpoint:

```
jcmd target/example-quarkus-SNAPSHOT.jar JDK.checkpoint
```

Restore:

```
$JAVA_HOME/bin/java -XX:CRaCRestoreFrom=cr
```

2.2.2. Сборка Micronaut приложения для OpenJDK CRaC

Теперь посмотрим на инструменты сборки такого же, приложения-примера, использующего Micronaut [6]. Представлены основные шаги сборки, целью которых является построение готового docker-образа. Docker [3] — программное обеспечение, позволяющее «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть развёрнут на любой Linux-системе. В готовом docker-образе находится язык разработки Java, Java приложение с нужными системными утилитами, библиотеками, настройками, а также образ checkpoint-a.

Создание образа:

```
./gradlew dockerBuildCrac
```

При исполнении данного шага, изначально запускается скрипт "checkpoint.sh"

Запуск образа:

```
$ docker run -d -p 8080:8080 hello:latest
```

В данный момент запускается bash-скрипт run.sh

Представлены описания bash-скриптов, запускаемых при создании, запуске docker-образа

1. checkpoint.sh

- запуск приложения, а именно в checkpoint режиме JDK
- дождаться старта

- исполнить `warmup.sh`
 - создать `checkpoint`
 - дождаться создания `checkpoint-a`
 - дать доступ `checkpoint-у` для хоста
2. `warmup.sh` — производит простой `curl` запрос у запущенного приложения для разогрева (хотелось бы, чтобы в будущем, тип запроса можно было конфигурировать, сделав его оптимальнее для текущего приложения).
 3. `run.sh` — `restore` с последнего чекпоинта, с `JAVA-OPTS`, которые можно указать при запуске образа.
 - в процессе сборки проекта, приложение запускается с помощью JVM в `checkpoint` режиме
 - после запуска проекта происходит разогрев на конфигулируемых данных
 - происходит `checkpoint`
 - `docker` образ CReC проекта создан

После запуска образа, происходит `restore` с последнего чекпоинта

Заключение

В ходе выполнения данной работы в осеннем семестре были достигнуты следующие результаты:

- сделать обзор проекта OpenJDK CRaC;
- произвести обзор инструментов сборки, существующих для других фреймворков, осуществляющих процедуру оптимизации приложений для OpenJDK CRaC, в частности, Micronaut.
- сформулировать требования к создаваемому шагу сборки;

Оставшиеся на весенний семестр задачи:

- реализовать шаг сборки по полученным требованиям, также сделав обзор шага сборки Quarkus для AppCDS [12];
- произвести апробацию решения;

Список литературы

- [1] Baeldung. AOT Usage. — <https://baeldung.com/ahead-of-time-compilation>. — 2022. — (дата обращения: 30.11.2022).
- [2] Belyaev Andrew. AOT With GraalVM Usage In Quarkus. — <https://habr.com/ru/company/haulmont/blog/487588/>. — 2022. — (дата обращения: 29.11.2022).
- [3] Docker. Docker Official Web Site. — <https://docker.com/>. — 2022. — (дата обращения: 29.11.2022).
- [4] GraalVM. GraalVM Official Web Site. — <https://graalvm.org/22.0/reference-manual/native-image/>. — 2022. — (дата обращения: 30.11.2022).
- [5] IBM. JIT-Compiler Explained. — <https://ibm.com/docs/en/ztpf/1.1.0.15?topic=reference-jit-compiler>. — 2022. — (дата обращения: 30.11.2022).
- [6] Micronaut. Micronaut Official Web Site. — <https://micronaut.io/>. — 2022. — (дата обращения: 29.11.2022).
- [7] OpenJDK. AppCDS Explained. — <https://openjdk.org/jeps/310>. — 2022. — (дата обращения: 29.11.2022).
- [8] OpenJDK. CRaC Github Project. — <https://github.com/CRaC/docs>. — 2022. — (дата обращения: 29.11.2022).
- [9] OpenJDK. OpenJDK Project Official Web Site. — <https://openjdk.org/>. — 2022. — (дата обращения: 29.11.2022).
- [10] Plizga Vladimir. AppCDS Slight Performance Increase. — <https://medium.com/@toparvion/appcds-for-spring-boot-applications-first-contact-6216db6a4194>. — 2022. — (дата обращения: 29.11.2022).

- [11] Price Mark. Why Does Java Apps Should Be Warmed-up. — <https://infoq.com/presentations/jvm-code-runtime/>. — 2022. — (дата обращения: 30.11.2022).
- [12] Quarkus. Example Quarkus App For AppCDS. — <https://github.com/gunnarmorling/quarkus-cds>. — 2022. — (дата обращения: 29.11.2022).
- [13] Quarkus. Example Quarkus App For CRaC. — <https://github.com/CRaC/example-quarkus#running>. — 2022. — (дата обращения: 29.11.2022).
- [14] Quarkus. Quarkus Benefits For Developers. — <https://quarkus.io/guides/>. — 2022. — (дата обращения: 30.11.2022).
- [15] Quarkus. Quarkus Official Github. — <https://github.com/quarkusio/quarkus>. — 2022. — (дата обращения: 30.11.2022).
- [16] Quarkus. Quarkus Performance Experiment. — <https://quarkus.io/blog/runtime-performance/>. — 2022. — (дата обращения: 29.11.2022).
- [17] TIOBE. The Official TIOBE Web Site. — <https://tiobe.com/tiobe-index/>. — 2022. — (дата обращения: 29.11.2022).
- [18] Tyson Matthew. What Is JDK. — <https://infoworld.com/article/3296360/what-is-the-jdk-introduction-to-the-java-development-kit.html>. — 2022. — (дата обращения: 29.11.2022).
- [19] Wagh Rahul. Example Quarkus Using Hot Reload. — <https://youtube.com/watch?v=8aZdcw9006g>. — 2022. — (дата обращения: 29.11.2022).
- [20] Wikipedia. Java Start-Up Performance Problems. — https://wikipedia.org/wiki/Java_performance#Startup_time. — 2022. — (дата обращения: 29.11.2022).