Оптимизация загрузки классов из кэша в виртуальной машине Java

Автор: Влаев Никита Владиславович, ОП ПИ 18.Б11-мм
Научный руководитель: д. ф.-м. н., доцент Д. В. Луцив
Консультант: ассистент кафедры системного программирования Козлов А.П.

Санкт-Петербургский государственный университет
Кафедра системного программирования

30 апреля 2022г.

JVM: холодный старт

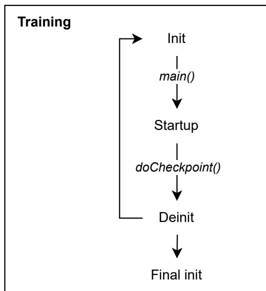
- Уходит много времени на старт
  - Загрузка системных классов
  - Загрузка классов приложения
  - Связывание
  - Инициализация классов
  - JIT-компиляция
- Особенно подвержены встроенные системы и IoT

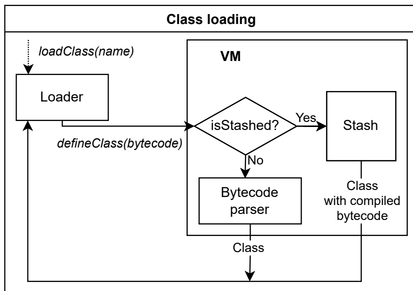
C/RaM

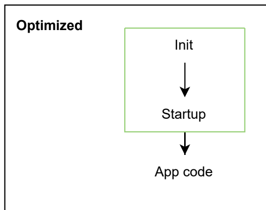
Checkpoint/Restore at Main – продукт компании Azul.

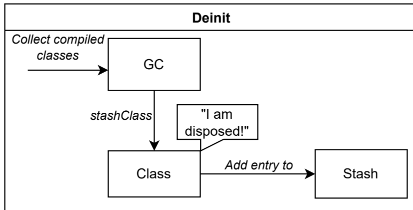
- Ускоряет инициализацию JVM
- Создается кэш классов и JIT-компиляций JVM, актуальный на момент старта приложения
- Для этого производятся тренировочные запуски приложения

C/RaM









C/RaM

В C/RaM JVM для доступа к записи кэша необходимо получить метаинформацию для байткода (значение CRC32, размер).

Проблема – Нужно *полностью загрузить класс* для получения полной метаинформации, а это лишняя нагрузка на файловую систему.

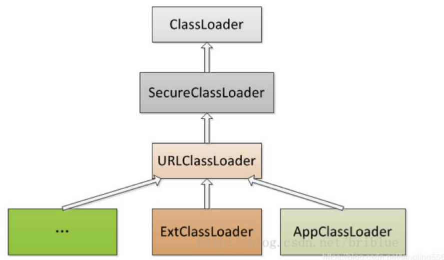
Постановка задачи

Целью данной работы является увеличение производительности кэширующей Java-машины за счет уменьшения объема считываемых с диска данных во время загрузки Java-классов.

Задачи

1. Сделать обзор предметной области:
   1. Процесс загрузки классов в Java
   2. Реализация кэша в C/RaM
   3. Возможные способы получения метаинформации для доступа к записи в кэше
2. Оптимизировать механизм загрузки классов JVM, реализовав способ обращения к кэшу без необходимости полной загрузки байткода класса
3. Провести тестирование производительности новой схемы загрузки Java-классов

Оптимизация системных загрузчиков классов



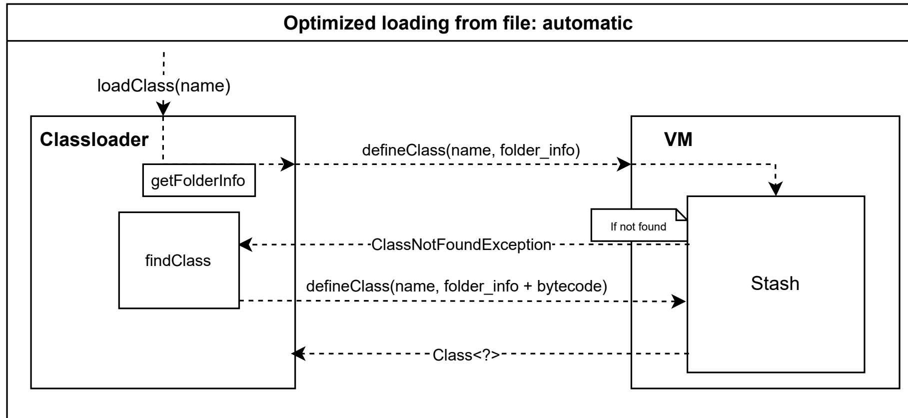
- Bootstrap – не нуждается
- Extension, App – используют URLClassloader
- URLClassloader – оптимизирован

Получение метаинформации для доступа к записи в кэше

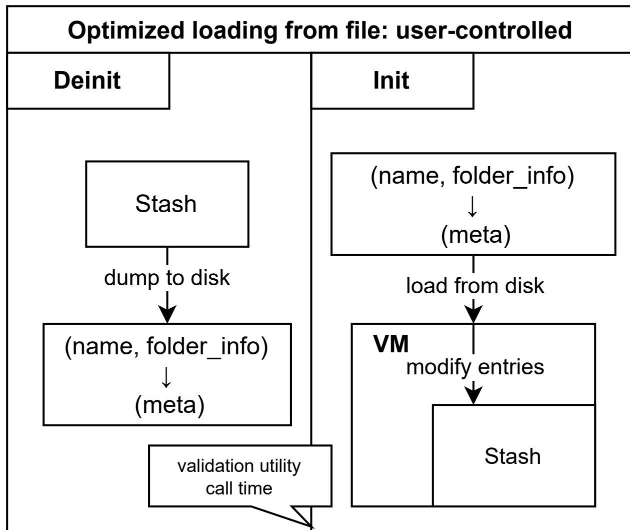
- Вычисление по байткоду напрямую
- Получение из Jar-файла
  - Хранится в записи Zip
- Сохранение во время тренировочного запуска
  - Может быть неактуальна
  - Нужен механизм валидации

Оптимизация URLClassloader

Получение метаинформации для файлов с байткодом:



Оптимизация URLClassloader



Оптимизация URLClassloader

Дополнен интерфейс java.lang.Classloader:

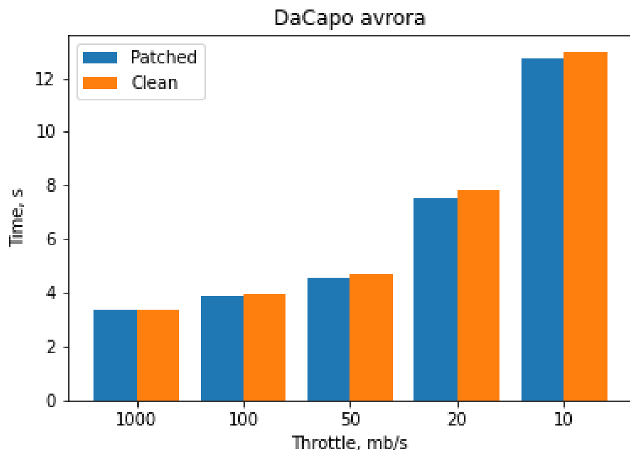
```
Class<?> defineClass(String name, java.nio.ByteBuffer b, CodeSource cs)
```

```
b = *header* + *CRC32* + *verification_metadata*
```

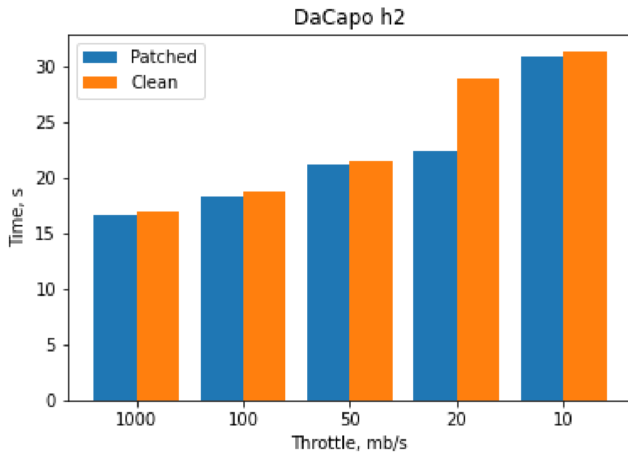
- Обновленный интерфейс описан в javadoc для использования пользовательскими загрузчиками

Апробация

Оборудование: Ubuntu 20.04, Intel core i7-6700 CPU, 3.4GHz, DDR4 16Gb RAM, SSD WDC PC SN730 SDBQNTY-512G-1001
DaCapo бенчмарк

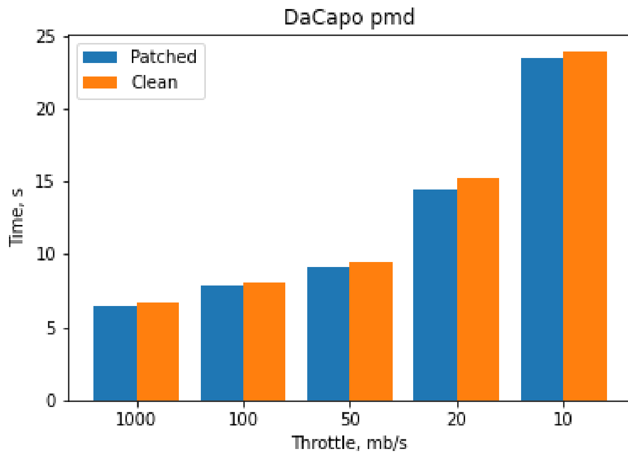


DaCapo бенчмарк



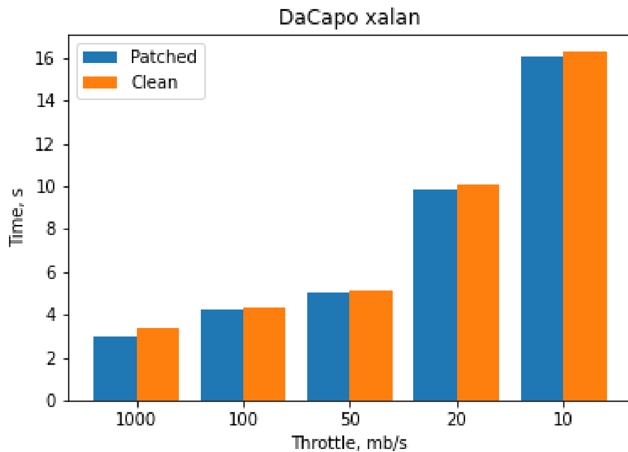
Апробация

DaCapo бенчмарк



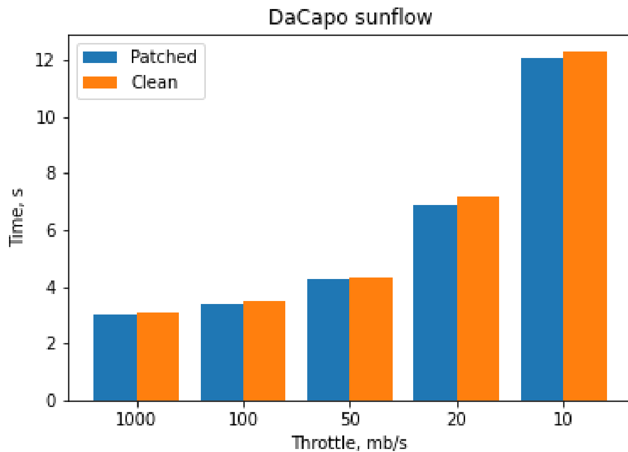
Апробация

DaCapo бенчмарк



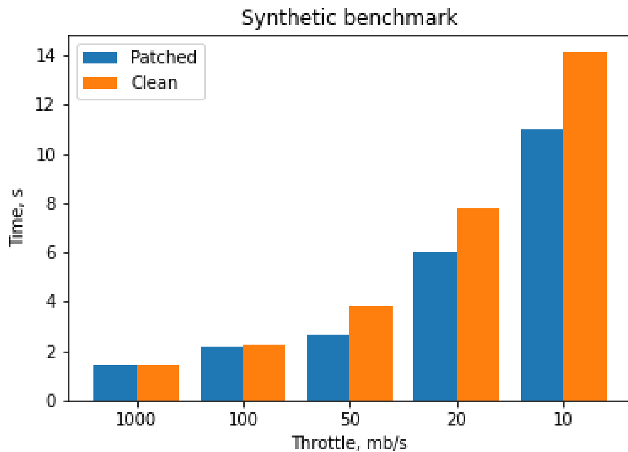
Апробация

DaCapo бенчмарк



Апробация

Синтетический бенчмарк: Google Guava, 2060 классов, 2.9мб



Результаты

1. Обзор предметной области
   - Процесс загрузки классов в Java
   - Реализация кэша в C/RaM
   - Способы получения метаинформации для доступа к записи в кэше
2. Оптимизирован механизм загрузки классов JVM:
   - URLClassloader
     - Получение метаданных из Jar-файлов
     - Сохранение метаданных на этапе тренировки для .class файлов
   - Описан интерфейс для пользовательских загрузчиков
   - Исследованы другие системные загрузчики: выявлено, что оптимизации не требуют
3. Проведено тестирование корректности и производительности
   - Синтетический бенчмарк: ускорен на 20% при ограничении доступа на диск в 10mb/s
   - DaCapo бенчмарк: ускорен в среднем на 2.5%(от 1.5% до 5%) при ограничении доступа на диск в 10mb/s