AXIOM JDK

Axiom NIK Pro
Выбор сборщика мусора (Garbage Collector)

Axiom NIK Pro | Март 2025

Программное обеспечение АКСИОМ содержит программное обеспечение с открытым исходным кодом. Дополнительная информация о коде сторонних разработчиков доступна на сайте https://axiomjdk.ru/third_party_licenses. Для дополнительной информации о том, как получить копию исходного кода, можно обратиться по адресу info@axiomjdk.ru.

ДАННАЯ ИНФОРМАЦИЯ МОЖЕТ ИЗМЕНЯТЬСЯ БЕЗ ПРЕДВАРИТЕЛЬНОГО УВЕДОМЛЕНИЯ. АКСИОМ ПРЕДОСТАВЛЯЕТ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ "КАК ЕСТЬ" БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, АКСИОМ ПРЯМО ОТКАЗЫВАЕТСЯ ОТ ВСЕХ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ И ПРИГОДНОСТИ ДЛЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ.

АКСИОМ НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ НЕ НЕСЕТ ОТВЕТСТВЕННОСТИ ЗА ЛЮБЫЕ КОСВЕННЫЕ, СЛУЧАЙНЫЕ, СПЕЦИАЛЬНЫЕ, ШТРАФНЫЕ ИЛИ КОСВЕННЫЕ УБЫТКИ, ИЛИ УБЫТКИ ОТ ПОТЕРИ ПРИБЫЛИ, ДОХОДА, ДАННЫХ ИЛИ ИСПОЛЬЗОВАНИЯ ДАННЫХ, ПОНЕСЕННЫЕ ВАМИ ИЛИ ЛЮБОЙ ТРЕТЬЕЙ СТОРОНОЙ, БУДЬ ТО В РЕЗУЛЬТАТЕ ДЕЙСТВИЯ ДОГОВОРА ИЛИ ДЕЛИКТА, ДАЖЕ ЕСЛИ АКСИОМ БЫЛО ПРЕДУПРЕЖДЕНО О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

Использование любого программного продукта АКСИОМ регулируется соответствующим лицензионным соглашением, которое никоим образом не изменяется условиями данного уведомления. Программные продукты и фирменные наименования: Axiom JDK, Axiom JDK Pro, Axiom Runtime Container Pro, Axiom Linux, Libercat, Libercat Certified и АКСИОМ принадлежат АКСИОМ и их использование допускается только с разрешения правообладателя.

Товарный знак Linux® используется в соответствии с сублицензией от Linux Foundation, эксклюзивного лицензиата Линуса Торвальдса, владельца знака на всемирной основе. Java и OpenJDK являются товарными знаками или зарегистрированными товарными знаками компании Oracle и/или ее аффилированных лиц. Другие торговые марки являются собственностью их соответствующих владельцев и используются только в целях идентификации.

Содержание

1. Введение в систему сборки мусора

Сборка мусора является важной и неизбежной частью JVM, которая влияет на общую производительность приложения. Полезно знать, какой сборщик мусора используется JVM, работающей в вашей среде Axiom NIK Pro.

Axiom NIK Pro предоставляет три сборщика мусора: **последовательный**, **параллельный** и **Epsilon** (неактивный).

- Последовательный сборщик мусора — это простой однопоточный алгоритм сборки мусора, который выполняет сборку мусора в одном потоке один за другим. Он подходит для небольших приложений или систем с низкими требованиями к памяти.
- Параллельный сборщик мусора использует несколько потоков для ускорения сборки мусора, что делает его более эффективным для больших приложений, работающих в многопроцессорной среде.
- Epsilon — это неактивный сборщик, который не собирает мусор. Он занимается только распределением памяти. Как только доступная память для Java будет исчерпана, JVM отключится.

Информация в этом документе актуальна для версии Axiom NIK Pro 24.0.1.

Важно отметить, что последовательный и параллельный сборщики мусора полностью функциональны в Axiom NIK Pro 24.0.1.

2. Параметры сборки мусора в NIK/Graal

Особенностью нативных образов, скомпилированных с помощью AOT, является то, что вы можете указать различные параметры, которые применяются на двух разных этапах создания и запуска этих образов. Одним из этапов является “сборка образа”, когда байт-код Java компилируется в машинный код. Другой - это “инициализация образа”, когда запускается собранный нативный образ.

Аналогично, параметры, управляющие поведением сборщика мусора, делятся на две категории. Некоторые из них применяются во время сборки образа и встраиваются в образ. Другие могут быть заданы во время инициализации нативного образа.

Чтобы получить полный список параметров нативного образа, выполните следующую команду:

```
native-image --expert-options-all
```

Параметры при сборке образа

Данные параметры передаются в утилиту `native-image` с использованием синтаксиса `-H:±Flag` или `-H:Setting=<значение>`.

Следующие параметры применяются к любому типу сборщика мусора во время создания образа.

Параметр	**Описание**
`-H:AlignedHeapChunkSize`	Выровненный размер фрагмента памяти.
`-H:HeapChunkHeaderPadding`	Число байтов в начале каждого фрагмента выделенной памяти, которые не используются для полезных данных, то есть байтов, которые могут свободно использоваться в качестве метаданных поставщиком фрагментов памяти.

Параметр	Описание
`-H:LargeArrayThreshold`	Размер выше которого массив выделяется в виде собственного невыровненного фрагмента.
`-H:±TreatRuntimeCodeInfoReferencesAsWeak`	Данный параметр определяет, следует ли рассматривать ссылки из скомпилированного во время исполнения кода на фрагменты памяти Java как сильные или слабые.
`-H:±VerifyHeap`	Проверять выделенную память до и после каждой сборки мусора.
`-H:±ZapChunks`	Заполнять неиспользуемые фрагменты памяти контрольным значением.
`-H:±ZapConsumedHeapChunks`	Перед использованием Заполнять фрагменты памяти контрольным значением.
`-H:±ZapProducedHeapChunks`	После использования Заполнять фрагменты памяти контрольным значением.

Следующие параметры применяются только к последовательному и параллельному сборщикам мусора во время создания образа.

Параметр	Описание
`-H:±CountWriteBarriers`	Инструментальные барьеры записи со счетчиками.
`-H:±GreyToBlackObjRefDemographics`	Составлять демографические данные о посещенных ссылках на объекты.
`-H:±IgnoreMaxHeapSizeWhileInVMOperation`	Игнорировать установленный максимальный размер выделенной памяти
`-H:±ImageHeapCardMarking`	Включает карточную разметку для объектов в выделенной памяти образа, что упорядочивает их по блокам. Автоматически включается, если поддерживается.
`-H:InitialCollectionPolicy`	Политика сборки мусора, либо Adaptive (по умолчанию), либо BySpaceAndTime.
`-H:MaxSurvivorSpaces`	Максимальное количество survivor spaces.
`-H:SoftRefLRUPolicyMSPerMB`	Количество миллисекунд, умноженное на свободное место в выделенной памяти в Мбайтах, представляет собой интервал времени, в течение которого слабая ссылка будет поддерживать активным свой референт после последнего доступа.
`-H:±VerifyAfterGC`	Проверять выделенную память после сборки мусора, если VerifyHeap включен.
`-H:±VerifyBeforeGC`	Проверять выделенную память перед сборкой мусора, если VerifyHeap включен.

Параметр	Описание
`-H:±VerifyReferences`	Проверять все ссылки на объекты, если VerifyHeap включен.
`-H:±VerifyReferencesPointIntoValidChunk`	Проверять, что ссылки на объекты ведут на валидные фрагменты памяти, если VerifyHeap включен.
`-H:±VerifyRememberedSet`	Проверять запомненный набор, если VerifyHeap включен.
`-H:±VerifyWriteBarriers`	Проверять барьеры записи.

Параметры при инициализации образа

Параметры инициализации передаются в исполняемый файл, созданный инструментом `native-image`, с использованием обычного синтаксиса Java: `-XX:±Flag` или `-XX:Setting=<value>`. Кроме того, вы можете указать их используя `-R:±Flag` или `-R:Setting=<значение>` (обратите внимание на отличие от параметров при сборке образа, которые используют `-H:`). Полученный двоичный файл компилируется с указанными вами значениями по умолчанию.

Следующие параметры применяются к любому типу сборщика мусора во время инициализации образа.

Параметр	Описание
`-H:±DisableExplicitGC`	Игнорировать вызовы `System.gc()`.
`-H:±ExitOnOutOfMemoryError`	Выходить из приложения при первом же возникновении ошибки нехватки памяти, если она возникает из-за нехватки памяти в выделенной памяти Java.

Параметр	Описание
`-H:MaxHeapSize`	Максимальный размер выделенной памяти во время выполнения приложения в байтах.
`-H:MaxNewSize`	Максимальный размер нового выделяемого фрагмента памяти во время выполнения, в байтах.
`-H:MaximumHeapSizePercent`	Максимальный размер выделяемой пмяти в процентах от физической памяти.
`-H:MaximumYoungGenerationSizePercent`	Максимальный размер нового выделяемого фрагмента памяти в процентах от максимального размера выделенной памяти.
`-H:MinHeapSize`	Минимальный размер выделенной памяти во время выполнения в байтах.
`-H:±PrintGC`	Распечатывать сводную информацию о сборщике мусора после каждой сборки мусора.
`-H:ReservedAddressSpaceSize`	Число байтов, которые следует зарезервировать для адресного пространства выделенной памяти.
`-H:±VerboseGC`	Выводить дополнительную информацию о выделенной памяти до и после каждой сборки мусора.

Следующие параметры применяются только к последовательному и параллельному сборщикам мусора во время инициализации образа.

Параметр	Описание
`-H:PercentTimeInIncrementalCollection`	Процент от общего времени сборки мусора, который должен быть потрачен на сборку мусора вновь создаваемых объектов, если используется политика сборки мусора `BySpaceAndTime`.
`-H:MaxHeapFree`	Максимальное количество свободных байт, зарезервированных для распределения памяти, в байтах (0 для автоматического в соответствии с политикой сборщика).
`-H:±TraceHeapChunks`	Отслеживание фрагментов выделенной памяти во время сборки мусора, если задано значение `+VerboseGC`.
`-H:±CollectYoungGenerationSeparately`	Определяет, собирает ли сборщик мусора вновь созданные объекты отдельно или вместе со старыми объектами.
`-H:±PrintGCSummary`	Распечатать сводную информацию о статусе сборщика мусора после возврата основного метода приложения.
`-H:±PrintGCTimes`	Вывести время для каждого этапа каждой сборки мусора, если задано значение `+VerboseGC`.

Данный параметр применяется исключительно к параллельному сборщику мусора во время запуска образа:

Option	Description
`ParallelGCThreads`	Количество рабочих потоков сборщика мусора.

3. Отслеживание использования памяти с помощью показателей JMX/JAR

Инструментарий Нативных Образов поддерживает JMX-интерфейсы MemoryMXBean и MemoryPoolMXBean. В этой статье объясняется, как их можно использовать для мониторинга памяти.

Поддерживаются следующие события JFR:

- jdk.AllocationRequiringGC
- jdk.GCHeapSummary
- jdk.GCPhasePause
- jdk.GCPhasePauseLevel
- jdk.GarbageCollection
- jdk.ObjectAllocationInNewTLAB
- jdk.ObjectAllocationOutsideTLAB
- jdk.ObjectCount
- jdk.ObjectCountAfterGC
- jdk.SystemGC

4. Получение и анализ данных журнала

Следующие команды и параметры помогут вам получать и анализировать журналы сборщика мусора.

Прежде чем приступить к анализу журналов, вы можете проверить тип сборщика мусора, используемого в вашей системе, выполнив команду `-Xlog:gc=info`.

- `-XX:+PrintGC` - Выдает данные журнала в формате аналогичном Java HotSpot, при использовании в imageruntime.
- `-XX:+VerboseGC` - Предоставляет более подробный вывод, показывающий, как каждый раунд сборки мусора повлиял на выделенную память.
- `XX:+PrintGCTimes` - Дополнительно показывает, как долго длится каждый этап работы сборщика мусора.

5. Выбор сборщика мусора для приложения

Важно:

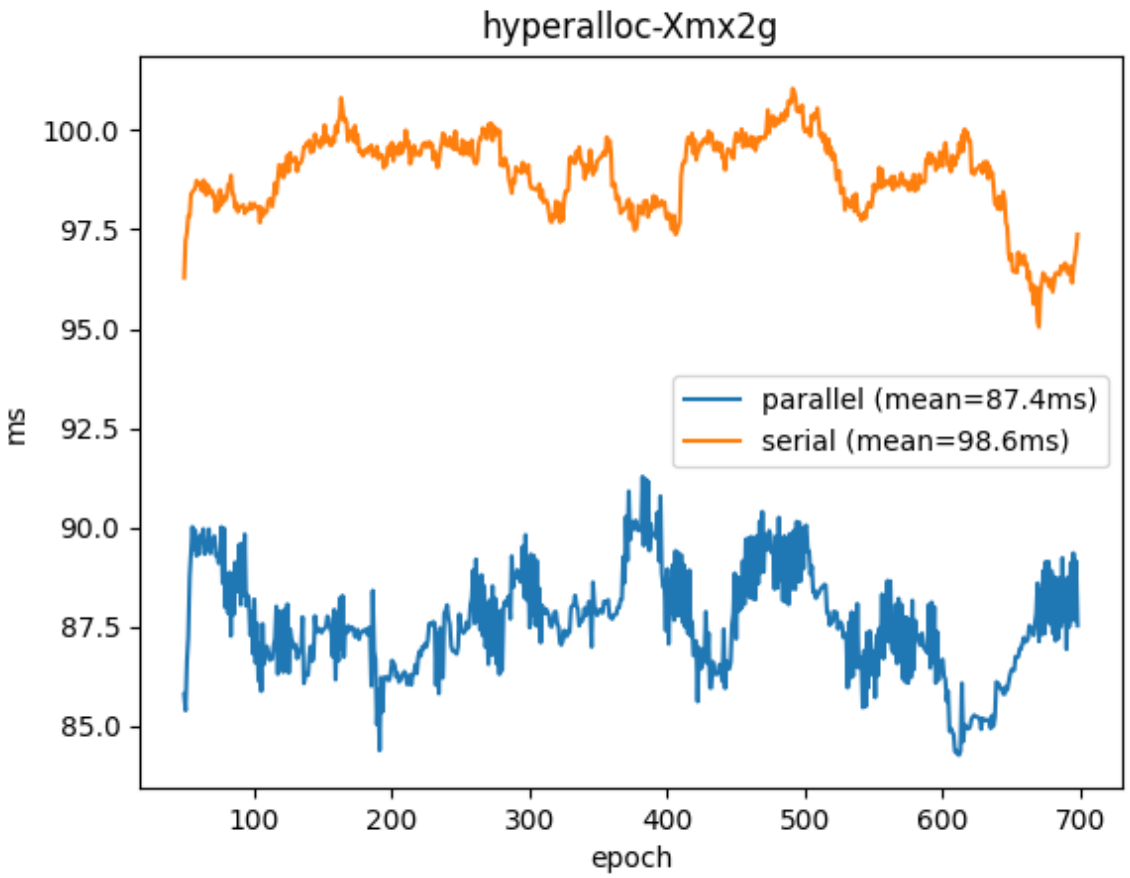
Последовательные и параллельные сборщики мусора готовы к работе в промышленных масштабах начиная с версии Axiom NIK Pro 24.0.0.

Производительность последовательной и параллельной сборки мусора

Меньшие паузы влияют на производительность приложения. Если для конкретного сервиса требуется как можно меньшее время отклика, последовательная сборка мусора - не ваш выбор.

Реализация параллельного сборщика мусора расширяет разнообразие типов сборщиков, доступных в Axiom NIK Pro, и позволяет улучшить время отклика.

Ниже на графике представлены время пауз, которое было измерено с помощью встроенной компиляции и запуска теста HyperAlloc benchmark из Проект Heapothesys. Цифры в таблице ниже - это время пауз при сборке мусора в миллисекундах. Тест был выполнен на Ubuntu, 8-ядерном процессоре i7 с 8 рабочими потоками и отключенной инкрементальной сборкой мусора.



Для получения дополнительной информации см. Parallel garbage collector.

Выбор сборщика мусора

Последовательный сборщик мусора (Serial GC) - старейший механизм сборки мусора, существующий с первых дней существования Java, и требует минимальных затрат. Он подходит для устройств с ограниченным объемом памяти и процессора, но в работе приложения могут возникать длительные паузы, особенно если задействован значительный объем памяти.

Если у вас большое приложение, работающее в многоядерной системе, мы рекомендуем использовать параллельный сборщик мусора (Parallel GC), поскольку он обеспечивает более низкое время паузы при достижении более высокой пропускной способности.

Сборщик мусора Epsilon GC полезен для измерения времени запуска программы, поскольку устраняет колебания, связанные с управлением памятью.

AXIOM JDK

Axiom NIK Pro
Выбор сборщика мусора (Garbage Collector)