Вопросы для подготовки:

- За какие функции отвечает JVM?

JVM выполняет две основные функции:

1) позволяет запускать Java-программы на любом устройстве или в любой операционной системе;

2) даёт доступ к управлению памятью программ и её оптимизации.

- Что такое Class Loader?

ClassLoader обеспечивает загрузку классов Java. Если говорить точнее, обеспечивают загрузку его наследники, конкретные загрузчики классов – сам ClassLoader абстрактен. Каждый раз, когда загружается какой-либо .class-файл, например, вследствие обращения к конструктору или статическому методу соответствующего класса – на самом деле это действие выполняет один из наследников класса ClassLoader.

Существует стандартный вариант реализации ClassLoader – так называемый системный загрузчик классов. Этот загрузчик используется по умолчанию при запуске приложений Java

- Какие функции выполняет Class Loader?

Загрузчики классов отвечают за динамическую загрузку классов Java в JVM (виртуальную машину Java) во время выполнения. Они также являются частью JRE (Java Runtime Environment). Следовательно, JVM не нужно знать о базовых файлах или файловых системах для запуска программ Java благодаря загрузчикам классов.

Кроме того, эти классы Java загружаются в память не сразу, а тогда, когда они требуются приложению. Здесь на помощь приходят загрузчики классов. Они отвечают за загрузку классов в память.

- Какие виды Class Loader'ов вы знаете?

Bootstrap, Extensions, System

- Расскажите про Bootstrap class loader

базовый загрузчик, также называется Primordial ClassLoader.

загружает стандартные классы JDK из архива rt.jar

- Расскажите про Extensions class loader

Загрузчик классов расширений является потомком загрузчика классов начальной загрузки и обеспечивает загрузку расширений стандартных основных классов Java , чтобы они были доступны для всех приложений, работающих на платформе.

Загрузчик класса расширения загружается из каталога расширений JDK, обычно это каталог $JAVA\_HOME/lib/ext или любой другой каталог, указанный в системном свойстве java.ext.dirs .

- Расскажите про System class loader

С другой стороны, загрузчик классов системы или приложения заботится о загрузке всех классов уровня приложения в JVM. Он загружает файлы, найденные в переменной среды classpath, -classpath или параметре командной строки -cp . Это также дочерний элемент загрузчика классов расширений.

- Расскажите про user-defined class loader

Однако в сценариях, когда нам нужно загружать классы с локального жесткого диска или из сети, нам может потребоваться использовать пользовательские загрузчики классов.

Пользовательские загрузчики классов полезны не только для загрузки класса во время выполнения. Несколько вариантов использования могут включать:

Помощь в изменении существующего байт-кода, например, ткацких агентов.

Создание классов, динамически подходящих для нужд пользователя, например, в JDBC переключение между различными реализациями драйверов осуществляется посредством динамической загрузки классов.

Реализация механизма управления версиями классов при загрузке разных байт-кодов для классов с одинаковыми именами и пакетами. Это можно сделать либо с помощью загрузчика классов URL (загрузка jar-файлов через URL-адреса), либо с помощью пользовательских загрузчиков классов.

Ниже приведены более конкретные примеры, где могут пригодиться пользовательские загрузчики классов.

Браузеры, например, используют пользовательский загрузчик классов для загрузки исполняемого содержимого с веб-сайта. Браузер может загружать апплеты с разных веб-страниц, используя отдельные загрузчики классов. Средство просмотра апплетов, которое используется для запуска апплетов, содержит ClassLoader , который обращается к веб-сайту на удаленном сервере вместо поиска в локальной файловой системе.

Затем он загружает необработанные файлы байт-кода через HTTP и превращает их в классы внутри JVM. Даже если эти апплеты имеют одинаковое имя, они считаются разными компонентами, если загружаются разными загрузчиками классов .

Теперь, когда мы понимаем, почему пользовательские загрузчики классов важны, давайте реализуем подкласс ClassLoader , чтобы расширить и суммировать функциональные возможности того, как JVM загружает классы.

Для его создания нужно расширить класс ClassLoader и переопределить метод findClass

- Расскажите про Run-Time Data Areas в JVM

Run-Time Data Areas - это области памяти, которые выделяются для хранения данных во время выполнения Java-программы. В Java существует несколько Run-Time Data Areas:

Method Area - область памяти, которая хранит описания классов, методов и других метаданных.

Heap - область памяти, которая хранит объекты, созданные во время выполнения программы.

Java Stack - область памяти, которая хранит данные локальных переменных и стек вызовов для каждого потока исполнения.

Native Method Stack - область памяти, которая хранит данные для вызова методов на языке, отличном от Java (например, C или C++).

PC Register - регистр, который содержит текущую инструкцию JVM для каждого потока исполнения.

gzE50U\_xa Memory - область памяти, которая используется для работы с прямой буферизацией данных.

Каждая из этих областей памяти имеет свои особенности и используется различными компонентами JVM во время выполнения программы.

Method Area содержит информацию о классах, интерфейсах, методах, полях и других метаданных. Эта область памяти разделяется между всеми потоками исполнения и не освобождается до завершения работы JVM.

Heap используется для создания и хранения объектов, которые создаются во время выполнения программы. Эта область памяти также разделяется между всеми потоками исполнения и автоматически управляется сборщиком мусора.

Java Stack содержит данные локальных переменных и стек вызовов для каждого потока исполнения. Каждый метод вызова имеет свой собственный фрейм данных в Java Stack.

Native Method Stack содержит данные для вызова методов на языке, отличном от Java (например, C или C++).

PC Register содержит текущую инструкцию JVM для каждого потока исполнения. Эта область памяти используется для управления потоками и переключения между ними.

gzE50U\_xa Memory используется для работы с прямой буферизацией данных. Эта область памяти не управляется сборщиком мусора и может быть освобождена только явным образом.

Таким образом, Run-Time Data Areas - это различные области памяти, которые выделяются для хранения данных во время выполнения Java-программы. Каждая из этих областей имеет свои особенности и используется различными компонентами JVM для выполнения своих функций.

- Что такое Run-Time Constant Pool?

таблица, в которой хранится дерево узлов, на которое разбиваются все классы и интерфейсы, аналог таблицы символов

- Расскажите о Frames

Frame используется для хранения данных и частичных результатов, а также для выполнения динамического связывания, возврата значений для методов и отправки исключений. Новый frame создается каждый раз, когда вызывается метод. Frame уничтожается, когда завершается вызов метода, является ли это завершение нормальным или резким (он генерирует неперехваченное исключение). Frames выделяются из стека потока, создающего frame. Каждый frame имеет свой собственный массив локальных переменных, свой собственный стек операндов и ссылку на пул констант во время выполнения класса текущего метода. Размеры массива локальных переменных и стека операндов определяются во время компиляции и предоставляются вместе с кодом для метода, связанного с фреймом. Таким образом, размер структуры данных, frame-а зависит только от реализации виртуальной машины Java, и память для этих структур может быть выделена одновременно при вызове метода.

Только один frame активен в любой точке данного потока управления - метода выполнения, и это frame называется текущим, а его метод известен как текущий метод. Класс, в котором определен текущий метод, является текущим классом. Операции над локальными переменными и стеком операндов обычно выполняются со ссылкой на текущий frame.

Frame перестает быть текущим, если его метод вызывает другой метод или если его метод завершается. Когда метод вызывается, новый frame создается и становится текущим, когда управление переходит к новому методу. При возврате метода текущий frame передает результат вызова метода, если таковой имеется, в предыдущий frame. Текущий frame затем отбрасывается, так как предыдущий frame становится текущим. Обратите внимание, что frame, созданный потоком, является локальным для этого потока и на него не может ссылаться ни один другой поток.

- Что такое JIT Compiler?

JIT-компиляция (англ. Just-in-Time, компиляция «точно в нужное время»), динамическая компиляция (англ. dynamic translation) — технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы. Таким образом достигается высокая скорость выполнения по сравнению с интерпретируемым байт-кодом[1] (сравнимая с компилируемыми языками) за счёт увеличения потребления памяти (для хранения результатов компиляции) и затрат времени на компиляцию. Технология JIT базируется на двух более ранних идеях, касающихся среды выполнения: компиляции байт-кода и динамической компиляции.

- GC и различные его виды в JVM.

Сборка мусора — это процесс восстановления заполненной памяти среды выполнения путем уничтожения неиспользуемых объектов.

Сборка мусора в Java — это процесс, с помощью которого программы Java автоматически управляют памятью.

В современных JVM есть следующие сборщики мусора: Serial Collector, Parallel Collector, Concurrent Mark Sweep (CMS) Collector и Garbage-First (G1) Collector. Они предлагают различные компромиссы между пропускной способностью, латентностью и использованием памяти, позволяя разработчикам выбрать наиболее подходящий сборщик для своего приложения.

- Какой объект считать достижимым. Как происходит сборка мусора?

Он начинает рассматривать набор корневых объектов — статические переменные, стек вызова и локальные переменные. Любой объект, на который непосредственно или косвенно ссылаются эти корневые объекты, считается достижимым и не подлежит сборке мусора.

- Расскажите о Serial GC

Serial GC: Использует однопоточную сборку мусора, подходит для небольших приложений с ограниченными ресурсами.

Parallel GC: Использует многопоточную сборку мусора для ускорения процесса, подходит для приложений с большим объемом памяти и многопроцессорными системами.

Concurrent Mark Sweep (CMS) GC: Осуществляет сборку мусора параллельно с работой приложения, уменьшая паузы, связанные с очисткой памяти. Подходит для приложений с высокими требованиями к отзывчивости.

G1 (Garbage First) GC: Высокопроизводительный алгоритм сборки мусора, который сосредоточен на минимизации пауз и управлении большими кучами памяти.

- Расскажите о Parallel GC

- Расскажите о Concurent Mark Sweep (CMS) GC

- Расскажите о Garbage-First (G1) GC

- Какие виды References вы знаете?

Для тех кто не знает есть 4 вида ссылок:

Strong reference

Weak Reference

Soft Reference

Phantom Reference

- Что такое Soft Reference?

Объекты, на которые указывают мягкие ссылки (SoftReference), не удаляются сборщиком мусора, пока память не начинает заканчиваться. При этом, если система сталкивается с нехваткой памяти, сборщик мусора может удалить эти объекты, чтобы освободить место.

Это полезно в случаях, когда нам будут нужны данные позже, и мы не хотим их вытеснять из памяти, если это не требуется. Например, это может быть полезно для кэширования данных.

- Что такое Weak Reference?

Объекты, на которые указывают слабые ссылки (WeakReference), удаляются при первой возможности сборщиком мусора, даже если памяти достаточно. После того как сборщик мусора удаляет объект, на который указывает слабая ссылка, она становится нулевой.

Слабые ссылки полезны, когда мы хотим иметь возможность получить доступ к объекту, но не хотим предотвратить его сборку мусора. Например, это может быть полезно для реализации некоторых структур данных, таких как кэши или мапы.

- Что такое Phantom Reference?

Фантомные ссылки возникают, когда объект уже признан мусором, финализирован и находится в процессе зачистки, о чём можно узнать с помощью класса Cleaner и выполнить в это время какие-то собственные действия.

- На какие области памяти разбит Heap?

Old generation и new generation

- Расскажите о Young Generation

Как следует из названия, Young Generation — это область памяти для новых, вновь создаваемых объектов.

Когда область Young Generation заполняется, то запускается минорная сборка мусора (Minor GC).

При Minor GC "мертвые" объекты удаляются из Young Generation.

Чем больше "мертвых" объектов в Young Generation, тем быстрее выполняется Minor GC.

При Minor GC происходит "остановка мира" (stop the world) — все потоки в приложении останавливаются.

Давайте подробнее разберемся со структурой Young generation.

Young generation разделен на три части: Eden, S0, S1.

Все новые объекты размещаются в Eden Space.

При заполнении Еden space происходит minor GC: все "живые" объекты перемещаются в одно из пространств уцелевших объектов (survivor space): S0 или S1. Допустим, в нашем случае, все объекты будут перемещены в S0.

- Расскажите о Old (Tenured) Generation

В Old generation располагаются долгоживущие объекты.

Как правило, эта область больше, чем область для young generation.

При заполнении (или достижении заданного порога) происходит Major GC.

Обычно Major GC выполняются медленнее и реже, чем minor GC.

- Расскажите о Permanent Generation

PermGen (Permanent Generation) — это специальное место в куче, отделенное от основной памяти.

В PermGen виртуальная машина хранит метаданные загруженных классов. Также здесь находятся всё статическое содержимое приложения, переменные примитивных типов и ссылки на статические объекты.

Более того PermGen хранит данные о байткоде и JIT информацию. До Java 7 пул строк также являлся частью PermGen.

По умолчанию, максимальный размер этой области памяти для 32-х битной JVM равен 64 Мб, а для 64-х битной версии — 82 Мб. Однако, вы можете изменять эти значения, используя следующие параметры JVM:

-XX:PermSize=[размер] для установки минимального размера PermGen области

-XX:MaxPermSize=[размер] для установки максимального размера

- Какие существуют ключи для настроек heap?

-Xmxsize (Maximum heap size) ...

-Xmssize (Minimum heap size) ...

-XX:NewSize=size (Young generation heap size) ...

-XX:MaxNewSize=size (Young generation max heap size) ...

-XX:+PrintGCDetails (Print GC details messages) ...

-XX:+PrintGCDateStamps (Print GC details messages)

- Что такое GraalVM?

Oracle GraalVM — это высокопроизводительный комплект разработчика JDK, который может ускорить работу Java и приложений на базе JVM с помощью альтернативного компилятора just-in-time (JIT). GraalVM Enterprise снижает задержки приложений, повышает пиковую пропускную способность за счет сокращения времени сборки мусора и обеспечивает круглосуточную поддержку Oracle.

Также предлагается утилита нативных образов, выполняющая АОТ-компиляцию байтового кода Java и генерирующая нативные исполняемые файлы для некоторых приложений, которые выполняются почти мгновенно и используют очень мало ресурсов памяти.

- Какие диагностические интерфейсы JVM вы знаете?