Конспект лекции
Classloaders

Цель и задачи лекции

Цель – изучить механизм работы загрузчиков классов.

Задачи:

1. Дать понятие classloader-а
2. Изучить принцип работы загрузчиков классов
3. Изучить виды загрузчиков классов

План занятия

1. Механизм загрузчиков классов
2. Виды загрузчиков классов
3. Принцип работы загрузчиков классов
4. Алгоритм работы загрузчиков классов
5. Пользовательские загрузчики классов
6. Выгрузка классов

Механизм загрузчиков классов

Загрузка представляет собой процесс поиска бинарного представления типа класса или интерфейса с определенным именем (возможно, вычисляемым “на лету”, но обычно получаемым путем выборки бинарного представления, ранее созданного компилятором Java из исходного текста) и построение на его основе объекта Class для представления класса или интерфейса.

Правильный загрузчик классов поддерживает следующие свойства.

- При получении одного и того же имени класса загрузчик должен возвращать один и тот же объект класса.
- Если загрузчик классов L1, делегирует загрузку класса C другому загрузчику L2, то для любого типа T, который является непосредственным суперклассом или суперинтерфейсом C, или типом поля в C, или типом формального параметра метода или конструктора C, или возвращаемым типом метода в C, L1 и L2, должны возвращать один и тот же объект Class.

Некорректный загрузчик классов может нарушать указанные правила. Однако он не может подорвать безопасность системы типов, поскольку виртуальная машина Java следит за этим.

Процесс загрузки

Процесс загрузки реализован классом ClassLoader и его подклассами.

Различные подклассы ClassLoader могут реализовывать различные стратегии загрузки. В частности, загрузчик классов может кешировать бинарные представления классов и интерфейсов, выполнять их предварительную выборку на основе ожидаемого применения или загружать группу связанных между собой классов. Эти действия могут не быть совершенно прозрачными для работающего приложения, если, например, вновь скомпилированная версия класса не найдена из-за того, что старая версия кеширована загрузчиком. Однако загрузчик классов отвечает за отражение ошибок загрузки только в тех точках программы, где они могут возникнуть без предварительной выборки или групповой загрузки.

Если ошибка возникает в процессе загрузки класса, то в любой точке программы, которая (прямо или косвенно) использует этот тип, генерируется исключение, представляющее собой экземпляр одного из следующих подклассов класса LinkageError.

Рассмотрим пример:

```
public void printClassLoaders() throws ClassNotFoundException {
    System.out.println("Classloader of this class:"
        + PrintClassLoader.class.getClassLoader());

    System.out.println("Classloader of Logging:"
        + Logging.class.getClassLoader());

    System.out.println("Classloader of ArrayList:"
        + ArrayList.class.getClassLoader());
}
```

В результате получим:

```
Class loader of this class:sun.misc.Launcher$AppClassLoader@18b4aac2
Class loader of Logging:sun.misc.Launcher$ExtClassLoader@3caeaf62
Class loader of ArrayList:null
```

Системный загрузчик классов загружает класс, в котором содержится метод в приведенном выше примере. Приложение или системный загрузчик классов загружает наши собственные файлы в classpath.

Затем загрузчик расширений загружает класс Logging. Загрузчики классов расширений загружают классы, являющиеся расширением стандартных базовых классов Java.

Наконец, bootstrap загрузчик загружает класс ArrayList. Bootstrap загрузчик классов является родителем всех остальных.

Тем не менее, мы видим, что последний выход, для ArrayList он отображает нуль в выводе. Это связано с тем, что загрузчик класса начальной загрузки написан на нативном

коде, а не на Java - поэтому он не отображается как класс Java. По этой причине поведение загрузчика классов начальной загрузки будет различным в разных виртуальных машинах.

Виды загрузчиков

Все классы в Java загружаются с помощью загрузчиков классов. Вначале работы программы создается 3 основных загрузчика классов:

- базовый загрузчик (bootstrap)
- системный загрузчик (system/application)
- загрузчик расширений (extention)

Любой класс (экземпляр класса java.lang.Class в среде и .class файл в файловой системе), используемый в среде исполнения был так или иначе загружен каким-либо загрузчиком в Java. Для того, чтобы получить загрузчик, которым был загружен класс A, необходимо воспользоваться методом A.class.getClassLoader().

Классы загружаются по мере надобности, за небольшим исключением. Некоторые базовые классы из rt.jar (java.lang.* в частности) загружаются при старте приложения. Классы расширений ($JAVA_HOME/lib/ext), пользовательские и большинство системных классов загружаются по мере их использования.

Bootstrap — реализован на уровне JVM и не имеет обратной связи со средой исполнения. Данным загрузчиком загружаются классы из директории $JAVA_HOME/lib. Т.е. всеми любимый rt.jar загружается именно базовым загрузчиком. Поэтому, попытка получения загрузчика у классов java.* всегда заканчиватся null'ом. Это объясняется тем, что все базовые классы загружены базовым загрузчиком, доступа к которому из управляемой среды нет.

Управлять загрузкой базовых классов можно с помощью ключа -Xbootclasspath, который позволяет переопределять наборы базовых классов.

System Classloader — системный загрузчик, реализованный уже на уровне JRE. В Sun JRE — это класс sun.misc.Launcher$AppClassLoader. Этим загрузчиком загружаются классы, пути к которым указаны в переменной окружения CLASSPATH. Управлять загрузкой системных классов можно с помощью ключа -classpath или системной опцией java.class.path.

Extension Classloader — загрузчик расширений. Данный загрузчик загружает классы из директории $JAVA_HOME/lib/ext. В JRE — это класс sun.misc.Launcher$ExtClassLoader. Управлять загрузкой расширений можно с помощью системной опции java.ext.dirs.

Различают также текущий загрузчик (Current Classloader) и загрузчик контекста (Context Classloader).

Current Classloader — это загрузчик класса, код которого в данный момент исполняется. Текущий загрузчик используется по умолчанию для загрузки классов в процессе исполнения. В часности, при использовании метода Class.forName("") и ClassLoader.loadClass("") или при любой декларации класса, ранее не загруженного.

Context Classloader — загрузчик контекста текущего потока. Получить и установить данный загрузчик можно с помощью методов Thread.getContextClassLoader() и Thread.setContextClassLoader(). Загрузчик контекста устанавливается автоматически для каждого нового потока. При этом, используется загрузчик родительского потока.

Принцип работы загрузчиков классов

Каждый загрузчик классов (кроме Bootstrap) имеет родительский загрузчик, и в большинстве случаев он запрашивает родительского загрузчика загрузить указанный класс, перед тем как попробовать загрузить его самостоятельно.

Существует так же явный способ инициировать загрузку требуемого класса. Явное инициирование выполняться с помощью методов ClassLoader.loadClass() или Class.forName(). Например явное инициирование используется при загрузке JDBC драйверов: Class.forName("oracle.jdbc.driver.OracleDriver").

Иерархия загрузчиков классов выглядит следующим образом:

1. Bootstrap
2. Extensions
3. Application
4. Пользовательский (если существует)

Чтобы получить загрузчик класса в коде, нужно воспользоваться методом getClassLoader(), например:

```
public class ClassLoadersTest {
   public static void main(String[] args){
        Integer i = 23;
        System.out.println(i.getClass().getClassLoader());
    }
}
```

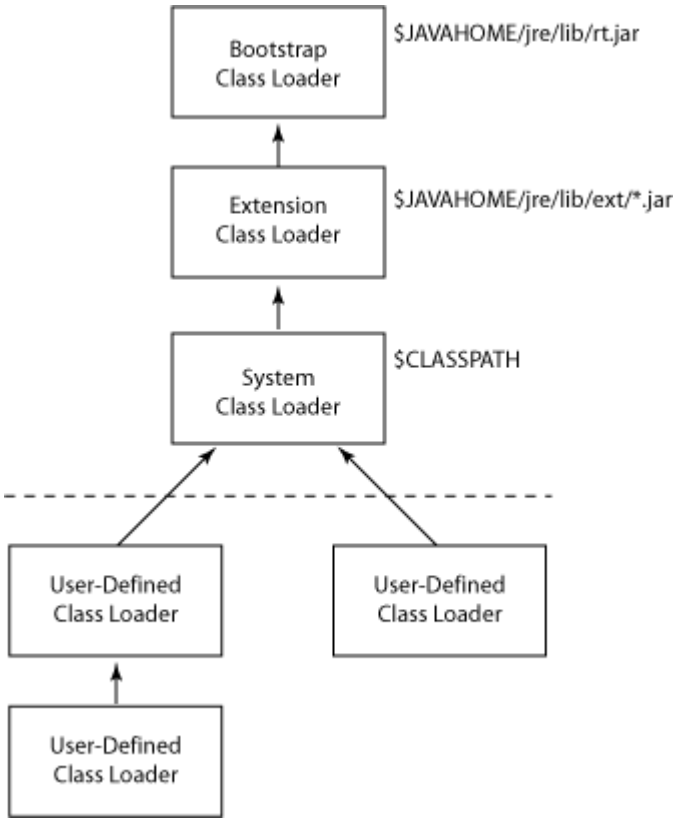
Вызов i.getClass().getClassLoader() вернет null, что свидетельствует о том, что класс был загружен именно базовым загрузчиком.

Рассмотрим процесс загрузки некоего пользовательского класса:

1. Системный загрузчик (sun.misc.Launcher$AppClassLoader) проверит, не загружался ли данный класс ранее. Если он уже загружался, то возвращается данный класс из кэша. Если нет, то системный загрузчик делегирует поиск класса родительскому классу-загрузчику.
2. Загрузчик расширений (sun.misc.Launcher$ExtClassLoader) выполняет такую же процедуру
3. Наконец, базовый загрузчик(bootstrap), загружает класс Integer самостоятельно, поскольку у него нет родительского класса.

Таким образом, процесс загрузки имеет одно важное свойство, а именно делегирование. Это позволяет загружать классы тем загрузчиком, который находится ближе всего к базовому в иерархии делегирования. Как следствие поиск классов будет

происходить в источниках в порядке их доверия: сначала в библиотеке core API, потом в папке расширений, потом в локальных файлах classpath.

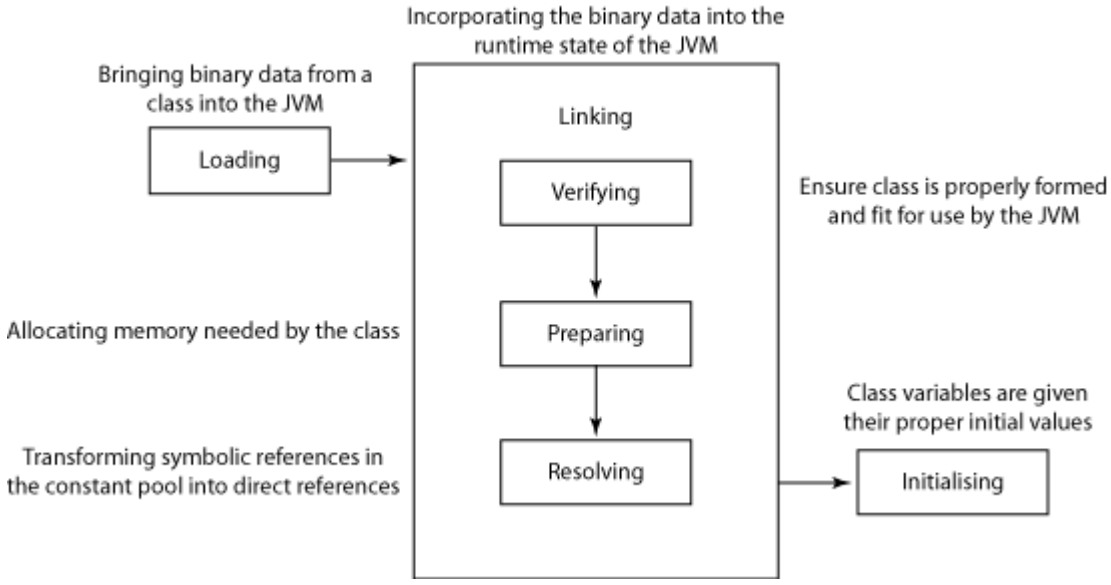


Стоит уточнить, что каждый загрузчик имеет свое пространство имен для создаваемых классов. Т.е. если классы одинаковы и находятся в одном пакете, но загружаются разными загрузчиками - они считаются разными.

Алгоритм работы загрузчика классов

Процесс загрузки класса состоит из трех частей:

1. Loading
2. Linking
3. Initialization



Loading – на этой фазе происходит поиск и физическая загрузка файла класса в определенном источнике (в зависимости от загрузчика). Этот процесс определяет базовое представление класса в памяти. На этом этапе такие понятия как методы, поля и т.д. пока не известны.

Linking – процесс, который может быть разбит на 3 части:

1. Bytecode verification – происходит несколько проверок байт-кода на соответствие ряду зачастую нетривиальных требований определенных в спецификации JVM (http://java.sun.com/docs/books/vmspec/).
2. Class preparation – на этом этапе происходит подготовки структуры данных, отображающей поля, методы и реализованные интерфейсы, которые определены в классе.
3. Resolving – разрешение все классов, которые ссылаются на текущий класс.

Initialization – происходит выполнение статических инициализаторов определенных в классе. Таким образом, статические поля инициализируются стандартными значениями.

Право загрузки класса рекурсивно делегируется от самого нижнего загрузчика в иерархии к самому верхнему. Такой подход позволяет загружать классы тем загрузчиком, который максимально близко находится к базовому. Так достигается максимальная область видимости классов. Под областью видимости подразумевается следующее. Каждый загрузчик ведет учет классов, которые были им загружены. Множество этих классов и называется областью видимости.

Рассмотрим процесс загрузки более детально. Пусть требуется загрузить объект пользовательского класс Student. В этом случае алгоритм будет следующий:

1. Системный загрузчик попытается поискать в кеше класс Student.

   1.1.Если класс найден, загрузка окончена.

   1.2.Если класс не найден, загрузка делегируется загрузчику расширений.

2. Загрузчик расширений попытается поискать в кеше класс Student.

   2.1.Если класс найден, загрузка окончена.

2.2.Если класс не найден, загрузка делегируется базовому загрузчику.

3. Базовый загрузчик попытается поискать в кеше класс Student.

3.1.Если класс найден, загрузка окончена.

3.2.Если класс не найден, базовый загрузчик попытается его загрузить.

3.2.1.Если загрузка прошла успешно, она закончена

3.2.2.Иначе управление предается загрузчику расширений.

3.3.Загрузчик расширений пытается загрузить класс.

3.3.1.Если загрузка прошла успешно, она закончена

3.3.2.Иначе управление предается системному загрузчику.

3.4.Системный загрузчик пытается загрузить класс.

3.4.1.Если загрузка прошла успешно, она закончена

3.4.2.Иначе генерируется исключение java.lang.ClassNotFoundException.

Пользовательские загрузчики классов

В Java существует возможность создания собственных загрузчиков классов. Это может быть полезно, когда нет возможности или нежелательно перечислять все используемые библиотеки при старте программы в CLASSPATH. Например, в программе должна быть возможность динамической загрузки плагинов. Или возможностей стандартного загрузчика недостаточно для загрузки нужных классов.

Собственные загрузчики классов используют все серверы приложений и web-контейнеры, что и понятно – приложения, разворачиваемые на сервере приложений, должны загружаться динамически, в противном случае перечисление в переменной CLASSPATH всех библиотек, используемых приложениями, становится задачей нетривиальной.

За создание пользовательских загрузчиков классов отвечает класс ClassLoader. Для того, чтобы создать собственный загрузчик классов, необходимо унаследоваться от класса ClassLoader.

Выгрузка классов

В большинстве случаев жизненный цикл класса в виртуальной машине схож с жизненным циклом объекта. JVM загружает, связывает и инициализирует классы, позволяя программе пользоваться ими, и выгружает, когда в приложении они более не используется. Важно заметить, что выгрузка классов не работает в том случае, если класс был загружен Bootstrap загрузчиком.

Загруженные классы, несмотря на то что являются полноценными Java-объектами, хранятся в особой системной области памяти, называемой permament generation (сокращенно, PermGen) и управляемой сборщиком мусора.

Выгрузка классов является важной частью механизма работы JVM, поскольку Java программы могут динамически расширяться во время работы, загружая пользовательские

классы и, тем самым, занимать много места в оперативной памяти. Держать классы в памяти, которые больше не будут использоваться, нет никакого смысла.

Конкретная политика выгрузки классов во многом зависит от реализации виртуальной машины JVM.

Литература и ссылки

1. Спецификация языка Java https://docs.oracle.com/javase/specs/jls/se8/html/index.html
2. Язык программирования Java SE 8. Подробное описание, 5-е издание, Джеймс Гослинг, Билл Джой, Гай Стил, Гилад Брача, Алекс Бакли; 672 стр., с ил.; 2015, 2 кв.; Вильямс.
3. Шилдт Г. Java 8. Полное руководство. 9-е издание. — М.: Вильямс, 2012. — 1377 с.
4. Эккель Б. Философия Java. 4-е полное изд. — СПб.: Питер, 2015. — 1168 с.
5. https://www.baeldung.com/java-classloaders

Вопросы для самоконтроля

1. Какие виды загрузчиков есть в Oracle JVM?
2. Опишите процесс загрузки пользовательского класса.