ЗАЩИЩЕННАЯ СРЕДА РАЗРАБОТКИ И ИСПОЛНЕНИЯ «JAVA AT»

Руководство пользователя (оператора)

Листов 29

2026

АННОТАЦИЯ

В данном программном документе приведено руководство пользователя (оператора) по применению и эксплуатации программного обеспечения «Защищенная среда разработки и исполнения «Java AT»» (далее – ПО «Java AT») для платформы X86, предназначенного для запуска Java программ в защищенной среде согласно требованиям ФСТЭК (второй уровня доверия).

В разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав программных средств).

В разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

В разделе «Сообщения пользователя (оператора)» указаны тексты сообщений, выдаваемых в ходе выполнения программы.

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ ПРОГРАММЫ

ПО «Java АТ» представляет собой совокупность программных средств и предназначено для разработки прикладных программ на языке Java и их последующего применения для обработки закрытых сведений категории «Для служебного пользования» (ДСП), а также сведений, составляющих государственную тайну. Программное обеспечение позволяет создавать, с использованием входящих в него инструментов и компонентов, прикладные программы различного назначения (серверные, клиентские части программ).

Структура ПО «Java АТ» позволяет применять его для разработки библиотек классов, автономных приложений, компонентов прикладных программ, а также программ, имеющих клиент-серверную архитектуру. Основной областью применения ПО «Java АТ» является создание прикладных программ серверной части, реализующих бизнес-логику, взаимодействие с системой управления базами данных (СУБД), смежными прикладными программами и ресурсами, сервисы для клиентской части программы и смежных прикладных программ.

Спецификой области применения ПО «Java АТ» является обеспечение соответствия требованиям информационной безопасности для выполнения прикладных программ, обрабатывающих информацию категории ДСП, а также информацию, составляющую государственную тайну вплоть до уровня «Совершенно секретно».

2. УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Для работы с ПО «Java АТ» пользователь должен обладать базовыми навыками работы с операционной системой, понимать базовые принципы программирования, знать основы синтаксиса языка Java, быть способным читать и анализировать сообщения об ошибках, возникающих при компиляции программ, обладать навыками работы с файлами (создание, редактирование и сохранение исходных кодов Java-файлов).

Желательно наличие базовых знаний английского языка. Рекомендуется заранее ознакомиться с базовыми учебными материалами по языку Java и принципам работы компиляторов.

2.1. Минимальный состав аппаратных средств

ПО «Java АТ» может быть установлено на персональный компьютер или сервер, требования к которым приведены далее. Минимальный состав используемых технических (аппаратных средств):

- персональный компьютер или сервер (Intel Core i3, ОЗУ 2 Гб, ПЗУ 256 Мб, Ethernet 10/100/1000);
- монитор 17” (1280x1024);
- клавиатура USB;
- манипулятор типа «мышь» USB.

2.2. Минимальный состав программных средств

Для работы ПО «Java АТ» необходима операционная система специального назначения «Astra Linux Special Edition» версии 1.6 или выше.

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

За исполнение программы, написанной на языке Java с использованием библиотек языка Java, отвечает модуль, именуемый виртуальной машиной Java или JVM. Данный модуль позволяет исполнить программу Java на любой из программных платформ, для которых он реализован.

Далее представлены основные команды для обращения к ПО «Java AT», описание их назначения и синтаксиса.

3.1. Команда java

Далее приведено описание команды java. Назначение команды:

- запуск Java-приложений;
- выполнение «*main*» - метода из указанного класса;
- управление параметрами виртуальной машины Java;
- запуск модульных приложений.

Синтаксис:

```
java [опции] <имя_класса> [аргументы]
java [опции] -jar <имя_jar_файла> [аргументы]
java [опции] --module <имя_модуля>/<имя_класса> [аргументы]
```

Основные опции команды java:

- запуск приложений;
- управление безопасностью;
- управление памятью;
- логирование и отладка;
- опции модульной системы;
- виртуальные потоки;
- прочие опции.

Синтаксис опции «запуск приложения»:

<имя_класса> - имя класса, содержащего метод public static void main(String[] args).

-jar <имя_jar_файла> - запуск приложения из JAR-файла, где указан главный класс в манифесте (Main-Class).

--module <имя_модуля>/<имя_класса> - запуск приложения из модуля (начиная с Java 9).

Синтаксис опции «управление безопасностью»:

-Dprotected.java.root=[absolute-path-1:…:absolute-path-N] - устанавливает набор абсолютных путей директорий (absolute-path-1,…, absolute-path-N), составляющих замкнутую программную среду для доверенного прикладного ПО.

--parsec-privsock - запуск виртуальной машины Java с возможностью создания привилегированных сокетов.

--parsec-changepriv - запуск виртуальной машины Java с возможностью создания привилегированного слушающего сокета, принимающего соединения с любыми мандатными метками.

--parsec-setmac <Уровень>:<Категория> - запуск виртуальной машины Java с заданной мандатной меткой

Синтаксис опции «управление памятью»:

-Xmx<размер> - максимальный размер кучи (например, -Xmx512m для 512 МБ).

-Xms<размер> - начальный размер кучи.

-Xss<размер> - размер стека для каждого потока.

-XX:G1PeriodicGCInterval=<время, мс> - период срабатывания гарантированной очистки памяти. По умолчанию 0 – периодическая очистка не вызывается. Работает только с сборщиком G1.

-XX:TriggerGCPeriod=<время, с> - период срабатывания гарантированной очистки памяти. По умолчанию 60 – вызывает сборщик мусора каждые 60 секунд. Эквивалент System.gc() – что позволяет работать с любым сборщиком мусора. Создает новый поток для вызова сборщика мусора на старте виртуальной машины.

Синтаксис опции «логирование и отладка»:

-verbose:class - вывод информации о загружаемых классах.

-verbose:gc - вывод информации о сборке мусора.

-Xlog:gc - расширенное логирование сборки мусора.*

-agentlib:<библиотека> - подключение агентов для профилирования или отладки (например, -agentlib:jdwp для отладки.

Синтаксис «опции модульной системы»:

--module-path <путь> или -p <путь> - указание пути к модулям.

--add-modules <модули> - добавление модулей для запуска.

--list-modules - вывод списка всех доступных модулей.

--describe-module <имя_модуля> - вывод информации о модуле.

Синтаксис опции «виртуальные потоки»:

-XX:+UseVirtualThreads - включение виртуальных потоков (легковесных потоков).

-XX:+PreserveFramePointer - оптимизация для виртуальных потоков.

Синтаксис «прочие опции»:

-D<property>=<value> - установка системных свойств (например, Dfile.encoding = UTF-8).

-version - вывод версии Java.

-help - вывод справки по команде java.

-cp <путь> или -classpath <путь> - указание пути к классам и ресурсам.

--enable-preview - включение предварительных функций (например, новых возможностей языка).

3.2. Команда javac

Далее приведено описание команды javac. Назначение команды:

- компиляция Java-кода в байт-код («*.class*» файлы);
- проверка кода на синтаксические и семантические ошибки;
- управление модульной системой при компиляции (начиная с Java 9);
- компиляция с поддержкой новых версий языка (включая preview-функции).

Синтаксис:

javac [опции] <имя_файла>.java

javac [опции] -d <путь_вывода> <имя_файла>.java

javac [опции] --module <имя_модуля> -d <путь_вывода> <имя_файла>.java

Основные опции команды java:

- компиляция;
- модульная система;
- оптимизация и контроль кода;
- предварительные функции и новые версии;
- прочие опции.

Синтаксис опции «компиляция»:

<имя_файла>.java - путь к исходному файлу, который нужно скомпилировать.

-d <путь> - указание каталога для вывода скомпилированных файлов.

-cp <путь> или -classpath <путь> - задание пути к зависимым классам и библиотекам.

-source <версия> - установка версии исходного кода (например, -source 17).

-target <версия> - установка версии байт-кода (например, -target 17)

Синтаксис опции «модульная система»:

--module-source-path <путь> - указание пути к исходникам модулей.

--module <имя_модуля>- компиляция модуля.

--module-path <путь> или -p <путь> - путь к зависимым модулям.

--add-exports <модуль>/<пакет>=<модули> - экспортирование пакетов в другие модули.

--add-opens <модуль>/ <пакет>=<модули> - открытие пакетов для рефлексии.

Синтаксис опции «оптимизация и контроль кода»:

-Xlint - включение предупреждений компилятора.

-Xlint:<тип> - выборочные предупреждения (unchecked, deprecation, cast, rawtypes).

-Werror - трактовать предупреждения как ошибки.

-parameters - сохранение имен параметров методов в байт-коде.

-g - добавление отладочной информации.

Синтаксис опции «предварительные функции и новые версии»:

--enable-preview - включение preview-функций языка.

--release <версия> - совместимость с определенной версией Java (например, --release 21).

Синтаксис «прочие опции»:

-verbose - подробный вывод компиляции.

-J<опция> - передача параметров в JVM при запуске компилятора.

-help - вывод справки по javac.

-version - вывод версии компилятора.

3.3. Команда jdb

Далее приведено описание команды jdb. Назначение команды:

– отладка Java-приложений на уровне байт-кода;

– установка точек останова и выполнение кода пошагово;

– анализ состояния потоков, переменных и стека вызовов;

– подключение к запущенной JVM для удаленной отладки.

Синтаксис:

jdb [опции] <класс>

jdb [опции] -jar <имя_jar_файла>

jdb -attach <адрес> (подключение к удаленной JVM)

jdb -connect <аргументы> (отладка через JDWP)

Основные опции команды «java»:

– запуск и подключение;

– управление точками останова;

– управление выполнением;

– анализ состояния программы;

– удаленная отладка;

– прочие опции.

Синтаксис опции «запуск и подключение»:

<класс> - класс с методом public static void main(String[] args), который будет запущен с отладкой.

-jar <имя_jar_файла> - запуск JAR-файла с отладкой.

-attach <адрес> - подключение к уже запущенной JVM (например, localhost:5005).

-connect com.sun.jdi.SocketAttach:hostname=<хост>,port=<порт> - подключение через JDWP.

-sourcepath <путь> - указание пути к исходникам для корректного отображения кода.

-classpath <путь> или -cp <путь> - указание пути к классам и зависимостям.

Синтаксис опции «управление точка останова»:

stop in <класс>.<метод> - установка точки останова в методе (stop in MyClass.main).

stop at <класс>:<номер_строки> - установка точки останова в конкретной строке (stop at MyClass:25).

clear <класс>:<номер_строки> - удаление точки останова.

clear - удаление всех точек останова.

list - просмотр кода вокруг текущей точки останова.

Синтаксис опции «управление выполнением»:

run - запуск программы.

cont - продолжение выполнения после остановки.

step - пошаговое выполнение (с заходом в методы).

next - пошаговое выполнение (без захода в методы).

finish - выполнение до выхода из текущего метода.

kill - завершение работы программы.

Синтаксис опции «анализ состояния программы»:

where - вывод стека вызовов текущего потока.

where all - вывод стеков всех потоков.

threads - просмотр списка потоков.

thread <номер> - переключение на другой поток.

print <выражение> - вывод значения переменной или выражения.

locals - просмотр локальных переменных текущего метода.

dump <объект> - вывод состояния объекта.

Синтаксис «удаленная отладка»:

JVM должна быть запущена с параметрами: java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=:5005 -jar myapp.jar*

Подключение jdb к удаленной JVM: jdb -attach localhost:5005

Синтаксис «прочие опции»:

help - вывод справки.

exit или quit - выход из отладчика.

! <команда> - выполнение системной команды

3.4. Команда javap

Далее приведено описание команды javap. Назначение команды:

- дизассемблирование «*.class*» файлов Java;
- просмотр структуры байт-кода, методов и полей класса;
- анализ компилированного кода без исходников;
- исследование модификаторов, аннотаций и сигнатур методов.

Синтаксис:

javap [опции] <имя_класса>

javap [опции] -cp <путь> <имя_класса>

javap [опции] -module <имя_модуля> <имя_класса>

Основные опции команды javap:

- просмотр структуры класса;
- анализ модульных классов.

Синтаксис опции «просмотр структуры класса»:

<имя_класса> - полное имя класса (например, com.example.MyClass).

-p - вывод всех полей и методов, включая private.

-v - детализированная информация о байт-коде (verbose).

-s - отображение дескрипторов полей и методов.

-c - дизассемблирование методов (байт-код JVM).

-constants - отображение констант пула строковых литералов.

-sysinfo - вывод системной информации о .class файле.

Синтаксис опции «анализ модульных классов»:

--module <имя_модуля> - анализ класса в модуле.

--module-path <путь> - указание пути к модулям.

3.5. Команда jar

Далее приведено описание команды jar. Назначение команды:

- создание, извлечение и управление архивами JAR (Java Archive);
- поддержка цифровой подписи и сжатия файлов;
- работа с модулями Java (начиная с Java 9).

Синтаксис:

jar [опции] <файл_jar> [файлы...]

jar --create [опции] --file <файл_jar> [файлы...]

jar --update [опции] --file <файл_jar> [файлы...]

jar --extract [опции] --file <файл_jar> [файлы...]

jar --list [опции] --file <файл_jar>

jar --describe-module --file <файл_jar>

Основные опции команды jar:

- создание JAR-файла;
- обновление JAR-файла;
- извлечение файлов из JAR;
- просмотр содержимого JAR.

Синтаксис опции «создание JAR-файла»:

--create (c) - создание нового JAR-файла.

--file <имя_jar> (f) - имя создаваемого архива.

--main-class <класс> - указание главного класса для запуска.

--manifest <файл> - указание пользовательского манифеста.

--compress - включение сжатия.

--verbose (v) - вывод информации о добавляемых файлах.

Синтаксис опции «обновление JAR-файла»:

--update (u) - добавление или обновление файлов в JAR.

--file <имя_jar> (f) - JAR-файл, который обновляется.

Синтаксис опции «извлечение файлов из JAR»:

--extract (x) - извлечение файлов из JAR.

--file <имя_jar> (f) - JAR-файл для распаковки.

--verbose (v) - вывод списка извлекаемых файлов.

Синтаксис опции «просмотр содержимого JAR»:

--list (t) - вывод списка файлов внутри JAR.

--file <имя_jar> (f) - JAR-файл, который нужно просмотреть.

3.6. Команда javadoc

Далее приведено описание команды javadoc. Назначение команды:

- генерация HTML-документации на основе комментариев javadoc в коде;
- поддержка тегов javadoc для описания классов, методов и полей.
- работа с модульной системой Java (начиная с Java 9).

Синтаксис:

javadoc [опции] <файлы_или_пакеты>

javadoc [опции] --source-path <путь> <пакеты>

javadoc [опции] --module <имя_модуля>

Основные опции команды javadoc:

- выбор исходных файлов для документации;
- настройка выходных данных;
- обработка тегов javadoc;
- форматирование и стиль;
- модульная система;
- фильтрация и отладка.

Синтаксис опции «выбор исходных файлов для документации»:

<файлы_или_пакеты> - список .java файлов или пакетов.

--source-path <путь> - путь к исходникам Java.

--module <имя_модуля> - генерация документации для модуля.

--module-source-path <путь> - указание пути к модулям.

--class-path <путь> (-classpath) - путь к зависимостям.

Синтаксис опции «настройка выходных данных»:

-d <каталог> – каталог для сохранения документации.

--frames – включение фреймов в HTML (по умолчанию).

--no-frames – отключение фреймов.

--quiet – скрытие предупреждений.

--verbose – вывод дополнительных логов.

Синтаксис опции «обработка тегов javadoc»:

--author - включение информации об авторе (@author).

--version - включение версии (@version).

--since - отображение информации о версиях (@since).

--tag <тег>:<формат> - кастомные теги.

Синтаксис опции «форматирование и стиль»:

--stylesheet-file <файл.css> - кастомные стили.

--docencoding <кодировка> - кодировка документации (по умолчанию UTF-8).

--charset <кодировка> - кодировка HTML-файлов.

Синтаксис опции «модульная система»:

--module <имя_модуля> - генерация документации для модуля.

--module-source-path <путь> - указание пути к исходникам модулей.

--add-modules <модули> - добавление модулей для обработки.

Синтаксис опции «фильтрация и отладка»:

--show-members <тип> - выбор уровня детализации (public, protected, package, private).

--show-packages <тип> - фильтр пакетов (all, exported).

--show-module-contents <тип> - что включать в документацию модуля (all, exported).

3.7. Команда keytool

Далее приведено описание команды keytool. Назначение команды:

- управление сертификатами и ключами в Java Keystore (JKS);
- создание, просмотр, импорт, экспорт и удаление ключей и сертификатов;
- работа с X.509-сертификатами и SSL/TLS.

Синтаксис:

keytool [опции]

Основные опции команды keytool:

- генерация ключевой пары;
- просмотр содержимого хранилища ключей;
- экспорт публичного ключа;
- импорт сертификата в хранилище доверенных сертификатов;
- создание запроса на сертификат;
- импорт подписанного сертификата в keystore;
- удаление ключа или сертификата;
- преобразование JKS → PKCS12.

Синтаксис опции «генерация ключевой пары»:

keytool -genkeypair -alias mykey -keyalg RSA -keysize 2048 -validity 365 -keystore keystore.jks

-genkeypair - создаёт ключевую пару.

-alias mykey - имя ключа.

-keyalg RSA - алгоритм (RSA, DSA, EC).

-keysize 2048 - размер ключа (1024, 2048, 4096).

-validity 365 - срок действия в днях.

-keystore keystore.jks - файл хранилища

Синтаксис опции «просмотр содержимого хранилища ключей»:

keytool -list -keystore keystore.jks

keytool -list -v -keystore keystore.jks - с детальной информацией

Синтаксис опции «экспорт публичного ключа»:

keytool -exportcert -alias mykey -file mycert.cer -keystore keystore.jks

-exportcert - экспортирует сертификат.

-file mycert.cer - имя выходного файла.

Синтаксис опции «импорт сертификата в хранилище доверенных сертификатов»:

keytool -importcert -alias trustedcert -file mycert.cer -keystore truststore.jks

-importcert - импортирует сертификат.

-keystore truststore.jks - хранилище доверенных сертификатов.

Синтаксис «создание запроса на сертификат»:

keytool -certreq -alias mykey -file mycertreq.csr -keystore keystore.jks

-certreq - создаёт CSR для центра сертификации (CA).

-file mycertreq.csr - имя выходного CSR-файла.

Синтаксис опции «импорт подписанного сертификата в keystore»:

keytool -importcert -alias mykey -file signedcert.cer -keystore keystore.jks

Синтаксис «удаление ключа или сертификата»:

keytool -delete -alias mykey -keystore keystore.jks

Синтаксис «преобразование JKS → PKCS12»:

keytool -importkeystore -srckeystore keystore.jks -destkeystore keystore.p12 -deststoretype PKCS12

Примеры использования:

Создать keystore с RSA-ключом (на 5 лет):
keytool -genkeypair -alias mykey -keyalg RSA -keysize 4096 -validity 1825 -keystore keystore.jks

Посмотреть все ключи в keystore:
keytool -list -keystore keystore.jks

Создать запрос на сертификат (CSR) для CA:
keytool -certreq -alias mykey -file mycertreq.csr -keystore keystore.jks

Импортировать сертификат CA в truststore:
keytool -importcert -alias rootCA -file ca-cert.cer -keystore truststore.jks

Экспортировать сертификат из keystore:
keytool -exportcert -alias mykey -file mycert.cer -keystore keystore.jks

3.8. Команда rmiregistry

Rmiregistry – это утилита для запуска реестра RMI (Remote Method Invocation), который используется для регистрации удалённых объектов и поиска их на удалённом сервере.

Синтаксис:

rmiregistry [nopm] – (необязательный параметр) на котором будет работать реестр. По умолчанию используется порт 1099

Основные опции команды rmiregistry:

- запуск RMI-реестра;
- запуск реестра из каталога.

Синтаксис опции «запуск RMI-реестра»:

Запуск rmiregistry без указания порта (реестр будет слушать на порту 1099.) *rmiregistry* С указанием порта (например, 2000): *rmiregistry 2000*

Синтаксис опции «запуск реестра из каталога»:

Например, если у вас есть классы в каталоге «/home/user/classes», выполните: *cd /home/user/classes rmiregistry*

Рекомендуется запускать rmiregistry в каталоге, где находятся скомпилированные классы (или хотя бы папка, где находятся их пути).

4. СООБЩЕНИЯ ОПЕРАТОРУ

В процессе работы программа может выдавать два типа сообщений: аварийные сообщения и сообщения об ошибках.

4.1. Аварийные сообщения

Аварийные сообщения информируют о том, что в процессе работы программы были обнаружены ошибки. Источником ошибок может быть, как некорректный текст исходной программы, так и неверные параметры запуска программы. В любом случае, работа программы будет аварийно завершена. При обнаружении аварийных сообщений программисту следует устранить причину ошибки и повторить запуск программы.

При некорректном завершении команды Java в стандартный вывод ошибок выводится сообщение об аварийном завершении работы и создается лог-файл с именем «*hs_err_pid<num>.log*», где «*num*» – идентификатор процесса Java.

Сообщение об ошибке внутри виртуальной машины, в поле «*fatal error*» выводится сигнал, который вызвал падение:

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
# Internal Error (os_linux_zero.cpp:274), pid=24451, tid=70369532072432
# fatal error: caught SIGILL at 0x000040002efe02f0
#
# JRE version: 6.0
# Java VM: OpenJDK 64-Bit Shark VM (19.0-b09 mixed mode linux-e2k )
# If you would like to submit a bug report, please include
# instructions how to reproduce the bug and visit:
#  http://icedtea.classpath.org/bugzilla
```

В лог-файле содержится следующая информация:

1) текущий поток;
2) стек фреймов Java машины;
3) список потоков на момент падения;

4) состояние памяти Java машины;
5) список загруженных динамических библиотек;
6) аргументы Java машины;
7) переменные среды окружения;
8) список обработчиков сигналов;
9) информация об операционной системе.

Пример лог – файла:

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
# Internal Error (os_linux_zero.cpp:274), pid=24451, tid=70369532072432
# fatal error: caught SIGILL at 0x000040002efe02f0
#
# JRE version: 6.0
# Java VM: OpenJDK 64-Bit Shark VM (19.0-b09 mixed mode linux-e2k )
# If you would like to submit a bug report, please include
# instructions how to reproduce the bug and visit:
#  http://icedtea.classpath.org/bugzilla
#

--------------- T H R E A D ---------------

Current thread (0x00004000301ccc60): JavaThread "Thread-1" [_thread_in_Java,
id=24717, stack(0x000040002ede8000,0x000040002ef68000)]

Stack: [0x000040002ede8000,0x000040002ef68000], sp=0x000040002eea80d8, free
space=768k
Java frames: (TOP FRAME MAY BE JUNK)

 0x000040002ef632c8: local[9]       = 0x00004000053231e0
 0x000040002ef632d0: local[8]       = 0x0000400027cf90e0
 0x000040002ef632d8: local[7]       = 0x000040002ef63298
 0x000040002ef632e0: local[6]       = 0x00004000053524f8
```

```
0x000040002ef632e8: local[5]       = 0x0000000000000003
0x000040002ef632f0: local[4]       = 0x0000400000000001
0x000040002ef632f8: local[3]       = 0x0000400000000000
0x000040002ef63300: local[2]       = 0x0000000000000005
0x000040002ef63308: local[1]       = 0x0000400009505720
0x000040002ef63310: local[0]       = 0x0000400009505528
0x000040002ef63318: istate->_bcp     = 0x0000400027d74a89 (bci 25)
0x000040002ef63320: istate->_locals    = 0x000040002ef633b0
0x000040002ef63328: istate->_constants  = 0x0000400027d76948
0x000040002ef63330: istate->_method    =
org.apache.derby.impl.store.access.heap.HeapController.lockRow(JIIZI)Z
0x000040002ef63338: istate->_stack    = 0x000040002ef632e8
0x000040002ef63340: istate->_msg     = 0x0000000000000008
0x000040002ef63348: istate->_callee    = 0x0000400027d74260
0x000040002ef63350: istate->_oop_temp   = 0x0000000000000000
0x000040002ef63358: istate->_stack_base  = 0x000040002ef63318
0x000040002ef63360: frame_type      = INTERPRETER_FRAME
0x000040002ef63368: next_frame      = 0x000040002ef63408
...
---------- P R O C E S S ----------
Java Threads: ( => current thread )
=>0x00004000301ccc60 JavaThread "Thread-1" [_thread_in_Java, id=24717,
stack(0x000040002ede8000,0x000040002ef68000)]
0x00004000302e53a0 JavaThread "derby.rawStoreDaemon" daemon [_thread_blocked,
id=24598, stack(0x000040002ec68000,0x000040002ede8000)]
0x000040003008e1f0 JavaThread "Timer-0" daemon [_thread_blocked, id=24585,
stack(0x000040002eae8000,0x000040002ec68000)]
0x000040003007ba30 JavaThread "derby.antiGC" daemon [_thread_blocked, id=24566,
stack(0x000040002e8e8000,0x000040002ea68000)]
0x0000400030058d30 JavaThread "Program Runner for derby" [_thread_blocked,
id=24464, stack(0x000040002e768000,0x000040002e8e8000)]
0x00000000000f64f0 JavaThread "Low Memory Detector" daemon [_thread_blocked,
id=24459, stack(0x000040002dce8000,0x000040002de68000)]
0x00000000000f2210 JavaThread "CompilerThread0" daemon [_thread_blocked, id=24458,
stack(0x000040002dbe9000,0x000040002dce8000)]
```

0x00000000000a4600 JavaThread "Signal Dispatcher" daemon [_thread_blocked, id=24457, stack(0x000040002d9e8000,0x000040002db68000)]

0x00000000000926f0 JavaThread "Finalizer" daemon [_thread_blocked, id=24456, stack(0x000040002d868000,0x000040002d9e8000)]

0x000000000008b860 JavaThread "Reference Handler" daemon [_thread_blocked, id=24455, stack(0x000040002d6e8000,0x000040002d868000)]

0x000000000002c870 JavaThread "main" [_thread_blocked, id=24453, stack(0x00004000050d0000,0x000040000524f000)]

Other Threads:

0x0000000000084c50 VMThread [stack: 0x000040002d526000,0x000040002d625000] [id=24454]

0x00000000000f79c0 WatcherThread [stack: 0x000040002de69000,0x000040002df68000] [id=24460]

VM state:not at safepoint (normal execution)

VM Mutex/Monitor currently owned by a thread: None

Heap

def new generation total 65472K, used 40740K [0x00004000072f0000, 0x000040000b9f0000, 0x0000400011d90000)

eden space 58240K, 60% used [0x00004000072f0000, 0x000040000953ca48, 0x000040000abd0000)

from space 7232K, 77% used [0x000040000b2e0000, 0x000040000b85c8b0, 0x000040000b9f0000)

to space 7232K, 0% used [0x000040000abd0000, 0x000040000abd0000, 0x000040000b2e0000)

tenured generation total 145288K, used 135955K [0x0000400011d90000, 0x000040001ab72000, 0x00004000272f0000)

the space 145288K, 93% used [0x0000400011d90000, 0x000040001a254f58, 0x000040001a255000, 0x000040001ab72000)

compacting perm gen total 14912K, used 14759K [0x00004000272f0000, 0x0000400028180000, 0x000040002d2f0000)

the space 14912K, 98% used [0x00004000272f0000, 0x0000400028159f70, 0x000040002815a000, 0x0000400028180000)

No shared spaces configured.

Dynamic libraries:

00002000-00023000 r-xp 00000000 08:03 1164348

/home/ikurdyukov/dist/bin/java
00023000-00025000 rwxp 00020000 08:03 1164348
/home/ikurdyukov/dist/bin/java
00025000-00e03000 rw-p 00000000 00:00 0 [heap]
400000000000-40000004d000 r-xp 00000000 08:03 393219
/lib64/ld-2.7.so
40000004d000-40000004f000 rwxp 0004c000 08:03 393219
/lib64/ld-2.7.so
40000004f000-400000051000 rw-p 00000000 00:00 0
400000051000-400000055000 r-xp 00000000 08:03 393236
/lib64/libdl-2.7.so
400000055000-400000056000 rwxp 00003000 08:03 393236
/lib64/libdl-2.7.so
400000056000-400000057000 rwxp 00000000 00:00 0
400000057000-400000084000 r-xp 00000000 08:03 393299
/lib64/libpthread-2.7.so
400000084000-400000087000 rwxp 0002c000 08:03 393299
/lib64/libpthread-2.7.so
400000087000-40000008b000 rwxp 00000000 00:00 0
40000008b000-4000000e6000 r-xp 00000000 08:03 742932
/usr/lib64/libcxa.so.2.0.0
4000000e6000-4000000e9000 rwxp 0005a000 08:03 742932
/usr/lib64/libcxa.so.2.0.0
4000000e9000-4000000ea000 rw-p 00000000 00:00 0
4000000ea000-4000003d8000 r-xp 00000000 08:03 393230
/lib64/libc-2.7.so
4000003d8000-4000003de000 rwxp 002ed000 08:03 393230
/lib64/libc-2.7.so
4000003de000-4000003e2000 rwxp 00000000 00:00 0
4000003e2000-400000468000 r-xp 00000000 08:03 393243
/lib64/libm-2.7.so
400000468000-40000046a000 rwxp 00085000 08:03 393243
/lib64/libm-2.7.so
...
VM Arguments:

jvm_args: -Xmixed

java_command: SPECjvm2008.jar -ict -icsv -crf 0 -ctf 0 -chf 0 -wt 240s -it 240s -bt 1 derby

Launcher Type: SUN_STANDARD

Environment Variables:

PATH=/opt/mcst/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/gtk2/bin:/usr/xfce4/bin:/usr/local/bin:/opt/mcst/stdp/bin

LD_LIBRARY_PATH=/home/ikurdyukov/dist/lib/e2k/server:/home/ikurdyukov/dist/lib/e2k:/home/ikurdyukov/dist/../lib/e2k:/lib:/usr/lib:/usr/X11R6/lib:/usr/gtk2/lib:/usr/xfce4/lib:/usr/qt3/lib

SHELL=/bin/bash

Signal Handlers:

SIGSEGV: [libjvm.so+0x2b3d68], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGBUS: [libjvm.so+0x2b3d68], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGFPE: [libjvm.so+0x28d7e8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGPIPE: [libjvm.so+0x28d7e8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGXFSZ: [libjvm.so+0x28d7e8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGILL: [libjvm.so+0x28d7e8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGUSR1: SIG_DFL, sa_mask[0]=0x00000000, sa_flags=0x00000000

SIGUSR2: [libjvm.so+0x28d4e8], sa_mask[0]=0x00000000, sa_flags=0x10000004

SIGHUP: [libjvm.so+0x28cfa8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGINT: [libjvm.so+0x28cfa8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGTERM: [libjvm.so+0x28cfa8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

SIGQUIT: [libjvm.so+0x28cfa8], sa_mask[0]=0xfffbfeff, sa_flags=0x10000004

---------- S Y S T E M ----------

OS:Linux

uname:Linux 2.6.33-elbrus.033.4.7.rt #1 SMP PREEMPT RT Sun Oct 27 21:36:27 MSK 2013 e2k

libc:glibc 2.7 NPTL 2.7

rlimit: STACK infinity, CORE 0k, NPROC 6403, NOFILE 1024, AS infinity

load average:4.34 5.54 5.93

/proc/meminfo:

MemTotal: 4099528 kB

MemFree: 412244 kB

Buffers: 325368 kB

Cached: 1352956 kB

SwapCached: 1888 kB

Active: 1281292 kB
Inactive: 835748 kB
Active(anon): 298932 kB
Inactive(anon): 140400 kB
Active(file): 982360 kB
Inactive(file): 695348 kB
Unevictable: 18800 kB
Mlocked: 18800 kB
SwapTotal: 2097144 kB
SwapFree: 2089756 kB
Dirty: 1872 kB
Writeback: 0 kB
AnonPages: 457076 kB
Mapped: 130432 kB
Shmem: 616 kB
Slab: 386272 kB
SReclaimable: 374424 kB
SUnreclaim: 11848 kB
KernelStack: 8400 kB
PageTables: 10212 kB
NFS_Unstable: 0 kB
Bounce: 0 kB
WritebackTmp: 0 kB
CommitLimit: 3622620 kB
Committed_AS: 1039756 kB
VmallocTotal: 1073741824 kB
VmallocUsed: 41112 kB
VmallocChunk: 1073695996 kB
HugePages_Total: 256
HugePages_Free: 256
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 4096 kB
CPU:total 2
Memory: 4k page, physical 4099528k(412244k free), swap 2097144k(2089756k free)

vm_info: OpenJDK 64-Bit Shark VM (19.0-b09) for linux-e2k JRE (1.6.0), built on Nov 20 2013 11:31:53 by unknown with gcc 4.4
time: Wed Nov 20 11:28:36 2013
elapsed time: 1648 seconds

4.2. Сообщения об ошибках работы программы

На формат и текст сообщений, выдаваемых виртуальной машиной в процессе работы программы, нет официального стандарта. Негласным стандартом для разработчиков стало их соответствие текстам и формату, выдаваемому оригинальной реализацией Java от Oracle. Реализация виртуальной машины Java для платформы X86 в своей основе имеет образ библиотеки классов и виртуальной машины из проекта OpenJDK, который в свою очередь основан на оригинальной виртуальной машине от Oracle. В силу этого формат всех сообщений, выдаваемых программисту реализацией машины Java для X86, полностью соответствует сообщениям, выдаваемым оригинальной виртуальной машиной, на любой из поддерживаемых ей платформ.

4.3. Сообщения пользователю от Java машины

Сообщения от Java машины имеют следующий формат:

Could not find main-class X in Y – не найден main метод.
Error occurred during initialization of VM. Could not reserve enough space for object heap. Could not create the Java virtual machine. – недостаточно памяти, чтобы инициализировать кучу для Java машины.
Local class incompatible: stream classdesc serialVersionUID = X - при сериализации был создан некорректный класс.
Your JAR-resources in JNLP-File are not signed from the same certificate. – JAR файл был неправильно подписан.
load: class X not found – загрузка несуществующего класса.

Сообщения об исключениях:

Exception in thread <Поток, в котором произошло исключение> <класс исключения>: <Опциональное сообщение исключения>
<Стек вызовов для текущего потока>

4.4. Сообщения пользователю от компилятора

Сообщения от компилятора имеют следующий формат:

<Имя файла>:<Номер строки>: error <Сообщение об ошибке>

Список сообщений об ошибках:

- *«public class should be in file»* – файл не содержит public класса;
- *«expected»* – недостающая точка с запятой в исходном тексте Java программы;
- *«possible loss of precision»* – неявная конвертация между примитивными типами;
- *«not accessible»* – попытка доступа к приватному полю или методу;
- *«not found in import»* – импортирование неверного пакета;
- *«type expected»* – попытка объявить переменную или поле без указания типа;
- *«can't be instantiated»* - попытка создания объекта интерфейса;
- *«{ expected»* – недостающая открывающая фигурная скобка;
- *«} expected»* – недостающая закрывающая скобка;
- *«unclosed String literal»* – некорректно построенная строка;
- *«missing method body»* – объявление метода без определения тела метода;
- *«case fallthrough»* – недостающий «break» в операторе выбора;
- *«undefined variable»* – использование необъявленной переменной;
- *«incompatible type»* – передача в метод объектов неверных типов;
- *«Exception never thrown»* – метод объявлен как кидающий исключение, но в теле метода нет операций, которые могут кинуть исключение;
- *«Unreported exception X: must be declared or caught»* – метод не объявлен как кидающий исключение, но в теле метода есть операции, которые могут кинуть исключение.

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

ДСП	Для служебного пользования
ПО «Java AT»	Программное обеспечение «Защищенная среда разработки и исполнения «Java AT»»
СУБД	Система управления базами данных
ФСТЭК	Федеральная служба по техническому и экспортному контролю
Astra Linux Special Edition	Защищённая операционная система Астра Линукс
JDK	от англ. Java Development Kit, комплект средств разработки для языка Java, включающий в себя компилятор, библиотеки классов и утилиты, необходимые для создания и компиляции Java-приложений
JKS	от англ. Java Keystore, формат хранения криптографических ключей и сертификатов.
JVM	от англ. Java Virtual Machine, виртуальная машина Java
PARSEC	Программное обеспечение для управления привилегиями пользовательских процессов
PKCS12	Стандартный формат файла для хранения сертификатов X.509 и соответствующих закрытых ключей, часто используемый для обмена и хранения криптографических ключей и сертификатов
RMI	от англ. Remote Method Invocation, механизм в Java, позволяющий объектам в одной виртуальной машине вызывать методы объектов, работающих в другой вирутальной машине, возможно, на другом компьютере