ЗАЩИЩЕННАЯ СРЕДА РАЗРАБОТКИ И ИСПОЛНЕНИЯ «JAVA AT»

Описание программы

Листов 18

2026

АННОТАЦИЯ

Данный документ является описанием программы программного обеспечения «Защищенная среда разработки и исполнения «Java AT»» (далее – ПО «Java AT») для платформы X86.

Описание программы состоит разделов, в которых раскрываются основные вопросы применения, структуры и функционирования программы.

В разделе «Общие сведения о программе» указаны наименование программы, сведения о технических и программных средствах, обеспечивающих выполнение данной программы, языки программирования, использованные при разработке программного обеспечения.

В разделе «Функциональное назначение программы» указано назначение и основные функции программного обеспечения (классы решаемых задач) программы.

В разделе «Описание логической структуры программы» общая структура программы.

В разделе «Используемые технические средства» описан необходимый вычислительный комплекс.

В разделе «Входные и выходные данные» приводятся общие сведения о входных и выходных данных.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

1.1. Обозначение и наименование программы

Наименование программы – ПО «Java AT».

1.2. Программное обеспечение, необходимое для функционирования программы

На вычислительном комплексе должна быть установлена операционная система Astra Linux SE версии 1.6 или выше.

В случае сборки из исходных кодов под ОС Astra Linux SE требуются следующие пакеты:

- виртуальная машина Java версии 20;
- установленные пакеты *«ant libfreetype6-dev», «libasound2-dev», «libffi-dev», «libx11-dev», «libxext-dev», «libxrender-dev», «libxtst-dev», «libxt-dev», «libcups2-dev», «pkg-config», «libfontconfig1-dev», «libelf-dev»*;
- архиватор zip.

1.3. Языки программирования, на которых написана программа

ПО «Java AT» написано на языке программирования C++.

Стандартные классы Java API написаны на языке программирования Java (Java).

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПРОГРАММЫ

ПО «Java АТ» предназначено для разработки и исполнения приложений, написанных на языке программирования Java. Язык Java – это высокоуровневый объектно-ориентированный язык общего назначения, опирающийся на поддержку стандартизованной среды исполнения (виртуальной машины и библиотек классов). Совокупность среды исполнения, библиотек классов и сопутствующих компонентов, а также спецификации языка Java, спецификаций виртуальной машины и интерфейсов к ней также называют платформой Java.

Структура ПО «Java АТ» позволяет применять его для разработки библиотек классов, автономных приложений, компонентов прикладных 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».

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ ПРОГРАММЫ

ПО «Java AT» предоставляет собой полную совместимую реализацию международной спецификации Java Standard Edition (Java SE) Platform версии 21. Основные характеристики:

- полная поддержка языка Java по стандарту Java Language Specification (JLS);
- полная реализация библиотек языка Java по стандарту Java SE 21 Edition;
- полная реализация виртуальной машины по стандарту Java Virtual Machine Specification (JVMS).

ПО «Java AT» включает в себя следующий набор программ:

- оптимизирующий компилятор javac;
- виртуальную машину Java;
- отладчик программ jdb;
- дизассемблер javap;
- программу архивации и сжатия файлов jar;
- программу генерации документации для программ на языке Java – javadoc;
- генератор пар «открытый/закрытый ключ» keytool;
- утилиту регистрации и вызова удаленных методов rmiregistry.

3.1. Виртуальная машина

Виртуальная машина Java (JVM, Java Virtual Machine) является ключевым компонентом платформы и основной частью ее исполнительной системы. JVM выполняет байт-код, сгенерированный компилятором Javac из исходного кода программы, путем интерпретации или использования динамической компиляции «на лету» (JIT, Just-In-Time Compiler).

Виртуальная машина Java может также использоваться для выполнения программ, написанных на других языках программирования.

JVM представляет собой программную среду, эмулирующую вычислительную машину с собственным набором инструкций, архитектурой и моделью исполнения.

3.2. Система управления памятью

Управление памятью в ПО «Java АТ» реализуется с помощью сборщика мусора (Garbage Collector, GC).

Основная задача сборщика мусора – своевременное выявление и освобождение участков памяти, занимаемых объектами, к которым отсутствует доступ из текущего состояния программы Java. В каждый определенный момент времени в программе присутствуют следующие типы ссылок, формирующие множество, называемое корневым («*root-set*»):

- ссылки в виде статических переменных;
- ссылки из временных переменных, находящихся на стеке каждого из потоков исполнения.

Все объекты, к которым можно получить доступ при переходе по ссылочным полям из объектов, входящих в корневое множество, являются «живыми». Все оставшиеся объекты являются «мертвыми» для сборщика мусора, и память, выделенная для них, высвобождается.

Существует множество вариантов и оптимизаций при реализации сборщика мусора.

По сравнению с ручным управлением памятью сборка мусора безопаснее, поскольку она предотвращает утечки памяти и возникновение висячих ссылок из-за несвоевременного удаления объектов. Другой положительный момент - упрощение самого процесса программирования.

Резервирование памяти также определяется виртуальной машиной, а не компилятором, либо же явным образом из программы. Разработчик может лишь указать, что он хочет создать еще один новый объект.

Указатели по физическим адресам отсутствуют принципиально.

3.3. Система безопасности Java

Важное свойство платформы Java – безопасность. Изначальная нацеленность на распределенные приложения сделала вопрос защиты одним из самых

приоритетных. При работе любой виртуальной машины Java для обеспечения безопасности действует целый комплекс мер.

3.4. Отсечение потенциально опасного кода

Потенциально опасный код отсекается на каждом этапе работы. Сначала байт-код загружается в систему, как правило, в виде класс-файлов. Виртуальная машина тщательно проверяет, все ли они подчиняются общим правилам безопасности и не искажены ли при передаче. Затем во время исполнения программы каждое действие с классом проверяется на допустимость. Возможности классов, которые были загружены с локального диска или по сети, существенно различаются (пользователь легко может назначать или отменять конкретные права).

В виртуальной машине Java реализована модель песочницы (sandbox) – изолированной среды, в которую помещаются программы (например, веб-приложения, полученные из потенциально небезопасных узлов). Песочница по умолчанию «отрезана» от файловой системы и может общаться только с тем узлом, с которого было загружено данное Java-приложение. Приложения, запущенные с локального диска, по умолчанию считаются безопасными, и им доступны все ресурсы виртуальной машины. Права доступа на имеющиеся ресурсы для каждой песочницы могут быть изменены в специальных конфигурационных файлах виртуальной машины.

3.5. Верификатор байт-кода Java

Верификатор байт-кода является неотъемлемой частью виртуальной машины. Он предназначен для проверки корректности структуры класс-файлов и объектной модели при загрузке классов. Верификатор обязательно запускается перед тем, как управление будет передано загруженному байт-коду.

Верификатор предотвращает следующие операции на уровне виртуальной машины:

– подделка указателей, например, получение указателя как результат выполнения арифметической операции;

- нарушение прав доступа к компонентам классов;
- вызов методов объектов с недопустимым набором параметров;
- вызов машинных инструкций с недопустимым набором параметров;
- некорректные операции с регистрами виртуальной машины;
- переполнение или исчерпание стека виртуальной машины.

Верификатор кода обеспечивает контроль границ объектов и массивов, делающий невозможным переполнение границ.

3.6. Менеджер потоков

Поток – это единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.

Поток представляется в виде объекта-потомка класса «*java.lang.Thread*». Этот класс инкапсулирует стандартные механизмы работы с потоком.

При запуске Java-программы операционная система создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса.

Тот поток, с которого начинается выполнение программы, называется главным. Выполнение главного потока начинается с метода «*main*». Затем, по мере необходимости, в заданных программистом местах, и при выполнении заданных им же условий, запускаются другие, побочные потоки. Таким образом, процесс может запускать несколько потоков, и каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.

Описанный механизм верен как для многопроцессорных многоядерных систем, так и для конфигураций с одним ядром процессора.

На одно ядро процессора, в каждый момент времени, приходится одна единица исполнения. Однако, запуск нескольких параллельных потоков возможен и в системах с одноядерными процессорами. В этом случае система будет периодически переключаться между потоками, поочередно давая выполняться

то одному, то другому потоку. Такая схема называется псевдо-параллелизмом. Система запоминает состояние (контекст) каждого потока, перед тем как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока.

В контекст потока входят такие параметры, как стек, набор значений регистров процессора, адрес исполняемой команды и прочее.

В виртуальной машине процесс завершается тогда, когда завершается последний его поток. Даже если метод «*main*» уже завершился, но еще выполняются порожденные им потоки, система будет ждать их завершения.

Однако, это правило не относится к особому виду потоков – демонам. Если завершился последний обычный поток процесса, и остались только потоки-демоны, то они будут принудительно завершены и выполнение процесса закончится.

Чаще всего потоки-демоны используются для выполнения фоновых задач, обслуживающих процесс в течение его жизни.

Менеджер потоков является посредником между средствами распараллеливания потоков, предоставляемыми операционной системой, и моделью потоков платформы Java.

Функции менеджера потоков:

- способствует локализации платформенной зависимости виртуальной машины Java;
- обеспечивает необходимые сервисы для работы сборщика мусора: обслуживает запросы на приостановку потоков, контролирует точки безопасной остановки;
- обеспечивает надежное и эффективное распараллеливание программ на многоядерных и многопроцессорных системах;
- предоставляет средства управления потоками, включая средства их создания и принудительного завершения;
- предоставляет средства для взаимодействия потоков друг с другом, включая примитивы синхронизации, средства прерывания потоков и поддержку приоритетности потоков;

- поддерживает дополнительные возможности для оптимизаций «на лету» на уровне потоков (тонкие/толстые мониторы, хелперы TLS);
- поддерживает инструментирование и инспектирование потоков для работы по протоколу JVMTI.

Процесс – это совокупность кода и данных, разделяющих общее виртуальное адресное пространство.

Для каждого процесса операционная система создает так называемое «виртуальное адресное пространство», к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. Операционная система же отвечает за то, как виртуальное пространство процесса проецируется на физическую память.

3.7. Интерфейс работы с нативными функциями JNI и реализация интерфейса Invocation API

Java Native Interface (JNI) – стандартный механизм, обеспечивающий взаимодействие ПО «Java AT» с кодом, написанным на языках C, C++ или ассемблере, под управлением виртуальной машины Java.

JNI позволяет вызывать функции, написанные на C или C++, из программ Java, а также создавать Java-объекты и вызывать их методы из кода на C или C++. Несмотря на то, что использование JNI снижает многоплатформенность кода Java, этот механизм расширяет возможности языка для приложений, где многоплатформенность не является обязательным требованием. JNI сочетает объектно-ориентированный подход Java с платформозависимым кодом, обеспечивая интеграцию с существующими системами.

JNI гарантирует двоичную совместимость для различных JVM на одной платформе, позволяя скомпилированному коду C или C++ выполняться в JVM, используемых в браузерах или средах разработки.

Платформозависимый интерфейс JNI предоставляет доступ к ограниченной части прикладного программного интерфейса (API) операционной системы.

В состав JNI входит Invocation API, обеспечивающий динамическую загрузку объектов JNI. Invocation API позволяет встраивать возможности Java в существующие системы без необходимости статического связывания с кодом JVM, выступая связующим звеном между ядром JVM и базовыми классами.

3.8. Интерпретатор

Интерпретатор является классической частью многих виртуальных машин. Цель интерпретатора – интерпретировать выполнение поданной на вход программы, состоящей из последовательности байт-кодов.

Плюсы использования интерпретатора:

- упрощение процесса разработки виртуальной машины, упрощенная отладка при разработке позволяет повысить качество продукта при сокращении сроков разработки;
- быстрый старт ряда приложений;
- упрощение реализации интерфейса для отладки, инструментирования байт-кода и мониторинга состояния виртуальной машины на низком уровне;
- возможность реализации техник частичной компиляции методов;
- минимальная зависимость от платформы.

3.9. Оптимизирующий динамический компилятор «на лету» (Just-in-time)

Компиляция «на лету» – технология увеличения производительности программных систем, использующих байт-код, путем компиляции байт-кода в машинный код непосредственно во время работы программы. Компилируются только те куски кода, про которые точно известно, что они будут исполняться. Таким образом достигается высокая скорость выполнения по сравнению с интерпретируемым байт-кодом (сравнимая с компилируемыми языками) за счет увеличения потребления памяти и затрат времени на компиляцию.

3.10. Библиотека классов

Платформа Java построена на основе стандартной библиотеки классов, которые осуществляют поддержку выполнения множества типовых задач, а именно:

- предоставляют общепринятый набор функций, аналогичный имеющимся в стандартных библиотеках, поставляемых с операционной системой (функции работы со строками, регулярными выражениями и коллекциями данных);
- определяют абстрактные программные интерфейсы для тех областей, которые обычно сильно зависят от аппаратной платформы или операционной системы;
- эмулируют или предоставляют механизмы проверки на наличие поддержки для того функционала, который присутствует только на ограниченном числе платформ (функции, обеспечивающие работу с графическим интерфейсом и особыми устройствами ввода/вывода данных).

Библиотека классов реализована преимущественно на языке Java, за исключением компонентов, требующих прямого доступа к системным функциям, таким как ввод-вывод или доступ к видеопамяти. Эти компоненты представляют собой обертки для функций библиотек операционной системы.

Библиотека классов Java включена в состав среды выполнения Java (JRE) и комплекта разработчика Java (JDK). При запуске приложений Java все классы библиотеки автоматически доступны, что обеспечивается платформой. Наличие нескольких JDK на одной системе не вызывает конфликтов, так как классы библиотеки корректно обнаруживаются платформой.

3.11. Подсистема связи виртуальной машины Java и библиотеки классов посредством базовых классов

В ПО «Java AT» базовые классы (kernel classes) обеспечивают взаимодействие между стандартной библиотекой классов и виртуальной машиной Java. Они представляют собой подмножество классов стандартной библиотеки, таких как

«*java.lang.Object*», «*java.lang.Class*» и другие, которые требуют низкоуровневого доступа к внутренним компонентам JVM или используются JVM особым образом.

Базовые классы состоят из нативной части и части, написанной на языке Java, взаимодействующих через Java Native Interface (JNI). Нативная часть связана с JVM посредством внутренних интерфейсов или является неотъемлемой частью ее исполняемых модулей. Это отличает базовые классы от обычных классов с нативными методами, которые не зависят от специфики конкретной JVM.

Наиболее значимые функции базовых классов:

- загрузка классов и доступ к классам;
- инспектирование стека;
- поддержка спецификации Java Reflection;
- поддержка ссылок и взаимодействие со сборщиком мусора, финализация объектов;
- поддержка процесса выгрузки классов;
- поддержка распараллеливания потоков и примитивов синхронизации (concurrency);
- «обертки» для доступа к прочим сервисам виртуальной машины;
- поддержка загрузки (bootstrap) и завершения работы (shutdown) виртуальной машины.

3.12. Обработчик сигналов/исключений

Обработчик сигналов и исключений необходим для полноценной обработки всех возможных ошибок исполнения, включая ситуации динамических ошибок типа исчерпания доступной динамической или стековой памяти, а также запросов на прерывание работы виртуальной машины от пользователя.

Он позволяет улучшить быстродействие скомпилированного байт-кода за счет использования низкоуровневых (аппаратных или системных) средств контроля исключений, а также предоставляет дополнительные сервисы для разработчиков виртуальной машины, включая средства для «посмертного» (post-mortem) анализа аварийного завершения машины.

3.13. Реализация интерфейса JVMTI

JVMTI – стандартный интерфейс для отладки, инструментирования байт-кода, и мониторинга виртуальной машины на низком уровне.

В силу своей универсальности он требует взаимодействия и поддержки от абсолютного большинства других компонентов виртуальной машины.

Не являясь обязательной частью виртуальной машины Java тем не менее, он существенно влияет на архитектурные и технические решения ключевых компонентов виртуальной машины.

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Для работы ПО «Java AT» необходима аппаратная часть, требования к которой приведены далее. Минимальный состав 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.

5. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ ПРОГРАММЫ

Входными данными для Java-машины являются текстовые файлы с исходным кодом (*.java*) или скомпилированные файлы байт-кода (*.class*), а также вводимые пользователем данные (текст или числа) с клавиатуры.

Выходными данными являются текстовые сообщения, отображаемые на консоли, записи в файлы (текстовые или бинарные), либо сообщения, передаваемые через сеть.

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

ДСП	Для служебного пользования
ПО «Java AT»	Программное обеспечение «Защищенная среда разработки и исполнения «Java AT»»
СУБД	Система управления баз данных
API	от англ. Application Programming Interface, программный интерфейс приложения
Astra Linux SE	Операционная система Астра Линукс
Invocation API	от англ. Invocation Application Programming Interface, программный интерфейс вызовов JNI
JAR	от англ. Java Archive, формат файлов для хранения классов и ресурсов Java
Java SE 21 Edition	Стандартная спецификация Java версии 21
JDK	от англ. Java Developer’s Kit, набор инструментов разработчика Java
JIT	от англ. Just-in-time, динамический оптимизирующий компилятор «на лету»
JLS	от англ. Java Language Specification, спецификации языка Java
JNI	от англ. Java Native Interface, нативный интерфейс Java
JRE	от англ. Java Runtime Environment, среда выполнения Java
JVM	от англ. Java Virtual Machine, виртуальная машина Java
JVMTI	от англ. Java Virtual Machine Tool Interface, инструментальный интерфейс виртуальной машины Java