AXIOM JDK

Axiom JDK Pro
Руководство пользователя

Axiom JDK I Февраль 2026

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

Настоящий документ содержит сведения по эксплуатации программного обеспечения «Axiom JDK Pro».

2. Режимы работы Axiom JDK Pro

Axiom JDK Pro предусматривает работу в двух режимах: режим разработки и режим выполнения, для сборки и отладки программного обеспечения и для выполнения готового кода соответственно.

Инструкции по работе с указанными режимами приведены в разделе Функции и интерфейсы функций Axiom JDK Pro.

3. Принципы безопасной работы Axiom JDK Pro

Для безопасного запуска программных компонент Axiom JDK Pro среда функционирования (операционная система) должна находиться исходно в безопасном состоянии. В случае если по не зависящим от Axiom JDK Pro причинам среда функционирования выходит из безопасного состояния, дальнейшее функционирование Axiom JDK Pro может быть нарушено. Таким образом считается, что компрометация среды исполнения приводит к автоматической компрометации всех ее компонент, в том числе программных компонент из состава Axiom JDK Pro.

На момент, предшествующий запуску программных компонент Axiom JDK Pro, в частности приложений Java под управлением среды исполнения Java из состава Axiom JDK Pro, все компоненты среды функционирования (операционной системы) полностью проинициализированы и находятся в безопасном состоянии.

Запуск программных компонент Axiom JDK Pro, в частности приложений Java под управлением среды исполнения Java из состава Axiom JDK Pro, производится авторизованным пользователем, использующим защищенный терминал и защищенный доверенный канал связи, имеющим соответствующие привилегии для запуска и остановки программных компонент Axiom JDK Pro.

Запуск программных компонент Axiom JDK Pro производится в контексте выделенной учетной записи, для которой среда функционирования (операционная система) не предоставляет привилегий записи на дисковые накопители (кроме выделенных областей, необходимых Java-приложению, исполняемому в контексте Axiom JDK Pro), таким образом процесс, содержащий Java-приложение, виртуальную машину и среду исполнения Java из состава Axiom JDK Pro, и запущенный в контексте выделенной учетной записи, не может изменить исполняемые модули и файлы конфигурации Axiom JDK Pro на дисковом накопителе. Таким образом предотвращается неавторизованное изменение исполняемых модулей и файлов конфигурации Axiom JDK Pro и нарушение целостности программной среды Axiom JDK Pro приложением или кодом, выполненным в контексте виртуальной машины Java из состава Axiom JDK Pro в результате атаки, компрометации приложения. Также предотвращается неавторизованное изменение и нарушение целостности среды функционирования (операционной системы) вследствие ограниченности привилегий выделенной учетной записи.

Запуск приложения Java под управлением среды исполнения Java из состава Axiom JDK Pro производится после полной инициализации среды исполнения Java из состава Axiom JDK Pro, включающей определение состояния конфигурации Axiom JDK Pro. Привилегии, с которыми осуществляется запуск приложения Java, не превышают привилегий выделенной учетной записи, с которыми произведен запуск программных компонент Axiom JDK Pro. Дальнейшее ограничение

привилегий реализуется механизмами менеджера безопасности, позволяющими производить более тонкую настройку правил доступа к ресурсам системы, доступным через посредство среды исполнения Java из состава Axiom JDK Pro. Политика безопасности, осуществляемая менеджером безопасности и установленная в файлах конфигурации Axiom JDK Pro администратором среды функционирования (операционной системы) определяет набор правил доступа к ресурсам (отдельным файлам, каталогам, сокетам, ресурсам внешних сетей, библиотечным Java-классам, специализированным действиям) кода приложения, полученного из источника (каталога, ресурса внешней сети), для которого установлены правила проверки целостности. Политика безопасности может определять (минимальный) набор привилегий для кода, для которого не установлен источник кода или не указаны правила проверки целостности, а также для динамически генерируемого и изменяемого средствами рефлексии Java-кода.

Изоляция памяти приложения Java, работающего под управлением среды исполнения Java из состава Axiom JDK Pro, достигается средствами среды функционирования (операционной системы), обеспечивающей изоляцию процессов в памяти.

Изоляция взаимодействия приложений через накопители обеспечивается применением выделенной учетной записи для процесса среды исполнения Java, в части операций записи, а также обеспечивается менеджером безопасности в части операций чтения и записи, в рамках ограничений выделенной учетной записи.

Изоляция сетевого взаимодействия обеспечивается менеджером безопасности посредством применения правил доступа к ресурсам сети и сокетам, заданных политикой безопасности для данного источника кода, а также применения правил маршрутизации пакетов, установленных в среде функционирования (операционной системе).

Защита функциональных возможностей, управляемых конфигурацией Axiom JDK Pro, от несанкционированного доступа обеспечивается запретом выполнения каких-либо действий до успешного прохождения процедуры идентификации и аутентификации и последующей авторизации средствами операционной системы, на которой установлен Axiom JDK Pro. Доступ к функциональным возможностям Axiom JDK Pro после успешной авторизации пользователя в операционной системе осуществляется в соответствии с правами доступа удостоверенного пользователя операционной системы.

4. Функции и интерфейсы функций Axiom JDK Pro

jar

Средство архивирования Java. Собирает множества файлов в единый архивный файл Jar.

Синтаксис

Параметры командной строки `jar` задаются в виде блока записанных слитно букв, которые передаются одним аргументом, а не через отдельные аргументы командной строки. Первая буква такого аргумента задаёт необходимое действие, которое должна выполнить программа jar.

`c` (create) - Создать новый JAR-архив. В качестве последних аргументов командной строки jar необходимо указать список файлов и/или каталогов.

`u` (update) - Обновить имеющийся JAR-архив. В качестве последних аргументов командной строки jar необходимо указать список файлов и/или каталогов.

`t` (table of contents view) - Вывести список файлов, содержащихся в JAR-архиве. Если задано имя JAR-файла с помощью параметра f, то список файлов выводится для него. В противном случае имя JAR-файла читается со стандартного устройства ввода.

`x` (extract) - Извлечь содержимое JAR-архива. Если задано имя JAR-файла с помощью параметра f, то извлекается содержимое этого файла. В противном случае имя JAR-файла читается со стандартного устройства ввода. Когда командная строка завершается списком файлов и/или каталогов, из JAR-архива извлекаются только файлы и каталоги, перечисленные в этом списке. В противном случае из архива извлекаются все файлы.

Вслед за идентификатором, определяющим выполняемое действие, могут следовать необязательные параметры:

`f` (file) - Указывает на то, что имя JAR-файла, который необходимо создать, из которого нужно извлечь файлы или получить список содержащихся файлов, задаётся в командной строке. Если f используется вместе с c, u, t или x, имя JAR-файла должно задаваться в качестве второго аргумента командной строки вызова y (т.е. оно должно располагаться непосредственно за блоком параметров). Когда этот параметр не задан, jar записывает создаваемый JAR-файл в стандартное устройство вывода или читает его со стандартного устройства ввода.

`m` (manifest) - Используется только в сочетании с параметром c и указывает на то, что jar должен использовать файл манифеста, указанный в командной строке, и использовать его в качестве основы для создания манифеста, который включается в JAR-файл. Когда этот параметр задаётся после параметра f, имя файла манифеста должно указываться после имени создаваемого архива. Если m стоит перед параметром f, то имя файла описания должно предшествовать имени файла создаваемого архива.

`v` (verbose) - Вывод имён обрабатываемых файлов. Если этот параметр задаётся вместе с c или u, то выводится имя каждого добавляемого или обновляемого в архиве файла со статистикой его сжатия. Когда параметр используется в сочетании с t, jar выводит список файлов, в котором кроме имени файла содержится его размер и дата последнего изменения. Если v указывается одновременно с x, то jar выводит имя каждого извлекаемого из архива файла.

Некоторые примеры приведены ниже:

Создание JAR-файла:

```
jar c[v0M]f jarfile [-C dir] inputfiles [-[J]_option_]
jar c[v0]mf manifest jarfile [-C dir] inputfiles [-[J]_option_] [-e
entrypoint]
jar c[v0M] [-C dir] inputfiles [-[J]_option_]
jar c[v0]m manifest [-C dir] inputfiles [-[J]_option_]
```

Обновление JAR-файла:

```
jar u[v0M]f jarfile [-C dir] inputfiles [-[J]_option_]
jar u[v0]mf manifest jarfile [-C dir] inputfiles [-[J]_option_] [-e
entrypoint]
jar u[v0M] [-C dir] inputfiles [-[J]_option_]
jar u[v0]m manifest [-C dir] inputfiles [-[J]_option_]
```

Распаковка JAR-файла:

```
jar x[v]f jarfile [inputfiles] [-[J]_option_]
jar x[v] [inputfiles] [-[J]_option_]
```

Вывод содержимого JAR-файла:

```
jar t[v]f jarfile [inputfiles] [-[J]_option_]
jar t[v] [inputfiles] [-[J]_option_]
```

Добавление индексов в JAR-файлы:

```
jar i jarfile [-[J]_option_]
```

Параметры

`inputfiles` - требуемые файлы или каталоги в команде разделяются пробелами при сжатии (c) или обновлении существующего архива (u), либо при извлечении (x) или получении списка файлов (t). Все каталоги обрабатываются рекурсивно. Файлы будут сжиматься, если не указан параметр -0 (ноль)

`manifest` - предварительно созданный файл манифеста MANIFEST.MF содержащий пары "ключ - значение" для дополнительных атрибутов (например, вендора ПО) добавляется в JAR-файл.

`entrypoint` - наименование класса, который назначается в качестве загрузчика приложений, реализованных в виде исполняемого JAR-файла.

`-C dir` - осуществляет временный (на период выполнения команды, аналог cd dir) переход в указанный каталог при обработке аргумента inputfiles. Допускается использовать несколько аргументов -C в одной команде.

`-[J]_option_` - передаёт опции загрузчика Java-приложений виртуальной машине Java. Например, `-J-Xms48m` устанавливает память запуска равной 48 мегабайтам.

Помимо стандартных опций доступно использование символов подстановки * и @ для перечисления объектов и ссылки на файл с перечнем объекта. Например, в примере ниже создаётся файл со списком классов, которые впоследствии упаковываются в архив `my.jar`:

```
C:\Java>  dir /b *.class > classes.list
C:\Java>  jar cf my.jar @classes.list
```

java

Компонент для запуска Java-приложений. Запуск приложений включает в себя старт JRE, загрузку указанного класса (см. entrypoint) и выполнения основного метода данного класса. Выполняемый при запуске метод должен быть объявлен как public и static, а также должен иметь возвращаемый тип void и должен иметь 1 входной параметр типа массив строк. Пример:

```
public static void main(String[])
```

По умолчанию первый не опциональный аргумент, является именем класса, который будет вызван. Должно использоваться имя класса с указанием полного пути к нему. Если указана опция `-jar`, то первым аргументом команды должно быть имя JAR-файла, содержащего класс и ресурсы

приложения, с загрузочным классом, определяемым параметром Main-Class манифеста.

JRE осуществляет поиск загрузочного и остальных классов в следующем порядке:

1. Системном (bootstrap) classpath (см. ниже).
2. В установленных расширениях ([jre]/lib/*.jar).
3. В пользовательском classpath.

Команда `javaw` идентична команде `java` за исключением того, что `javaw` не связано с окном консоли. `javaw` используется в случаях, когда не нужно, чтобы появлялось окно командной строки. Синтаксис команды java следующий:

```
java [ options ] class [ arguments ]
java [ options ] -jar file.jar [ arguments ]
javaw [ options ] class [ arguments ]
javaw [ options ] -jar file.jar [ arguments ]
```

Опции

- `class` - наименование вызываемого класса.
- `file.jar` - наименование вызываемого JAR-файла. Может быть использовано только совместно с командой -jar.
- `arguments` - аргументы основной функции.

Параметры

`-client`/`-server` - выбор клиентской или серверной модификаций JVM

`-javaagent` - загрузка Java-агента. javaagent это один из параметров JVM, который позволяет указать агент который будет запущен перед стартом приложения. Сам агент - это отдельное приложение которое предоставляет доступ к механизму манипуляции байт-кодом (java.lang.instrument) во время выполнения.

`-cp`(`-classpath`) - указание путей, по которым производится поиск классов, необходимых для запуска. Различные пути разделяются между собой точкой с запятой (;) или двоеточием (:) в ОС Windows/Linux соответственно. Указанием -classpath перезаписывается переменная окружения CLASSPATH. Если не указаны переменная окружения CLASSPATH и параметр -classpath, то поиск класса осуществляется только в текущей директории.

`-jar` - выполняет программу из JAR-файла. Первый аргумент является именем JAR файла и

выполняет указанный в манифесте загрузочный класс.

`-agentlib:libname[=options]` - загрузка отладочного агента, например:

```
 -agentlib:hprof -agentlib:jdwp=help agentlib:hprof=help.
```

Команда `-agentlib:foo=opt1,opt2` означает, что Windows должен найти и загрузить — `foo.dll`, а OS Linux — `libfoo.so`.

`-[D]_property_="value"` - установка системных свойств. Пример вызова:

```
 java -Dmydir="some  string" SomeClass
 -enableassertions :… |:
 -ea :… |:
```

Включает диагностические проверки, которые по умолчанию выключены. Диагностические проверки осуществляют контроль данных в методах и, если некие данные не удовлетворяют условию проверки, происходит принудительное завершение приложения. Если не указаны аргументы, то диагностические проверки будут включены во всём приложении, в противном случае только в указанных пакетах или классах. Если задать один пакет и значение "…", то диагностические проверки будут включены в указанном пакете и всех подпакетах. Если указать значение "…", то диагностические проверки будут включены во всех пакетах, найденных в текущей директории. Пример:

```
 java -ea:com.wombat.fruitbat... <Main Class>
```

`-disableassertions :<package name>… |:<class; -da :<package name>… |:<class name>` - отключает диагностические проверки.

`-enablesystemassertions` / `-esa` - включает диагностические проверки в системных классах

`-disablesystemassertions` / `-dsa` - отключает диагностические проверки в системных классах

`-javaagent:jarpath[=options]` - загрузка агента Java. См. выше.

`-jre-restrict-search` - флаг включения пользовательских сред выполнения Java в поиск версии командой `-version` или `-showversion`.

`-no-jre-restrict-search` - флаг исключения пользовательских сред выполнения Java.

`-showversion` - отображение информации о версии и продолжение работы.

`-splash:)imagepath` - отображение на экране изображения, расположенного в указанном в параметре пути.

`-version` - отображение информации о версии и завершение работы.

`-version:release` - вывод информации о версии указанных релизов.

`-verbose:class` - отображает информацию о каждом загруженном классе.

`-verbose:gc` - отображает отчёт по сборщику мусора (garbage collection event).

`-verbose:jni` - информация об используемых нативных методах или других Java Native Interface.

Расширенные параметры

`-X` - ключ используемых расширенных параметров.

`-Xdebug` - запуск приложения в режиме отладки

`-Xint` - действовать только в режиме интерпретатора. Компиляции в native код производиться не будет, и байткод будет выполняться только в режиме интерпретатора. В этом режиме преимущества JIT-компилятора Java HotSpot Server VM не будет использоваться.

`-Xbootclasspath:bootclasspath` - указание списка каталогов, JAR-файлов или ZIP-архивов для поиска загрузочных классов. Список указывается через точку с запятой.

`-Xbootclasspath/a:path` - указание списка путей к каталогам, JAR-файлам или ZIP-архивам для поиска загрузочных классов после (append) класса по умолчанию. Список разделяется точкой с запятой (;) или двоеточием (:) в ОС Windows/Linux соответственно.

`-Xbootclasspath/p:path` - указание списка путей к каталогам, JAR-файлам или ZIP-архивам для поиска загрузочных классов перед (prepend) классом по умолчанию. Список указывается через точку с запятой.

`-Xcheck:jni` - дополнительная проверка для функций Java Native Interface (JNI). В частности, виртуальная машина Java проверяет параметры, переданные функции JNI, а также данные среды выполнения перед обработкой запроса JNI. Любые обнаруженные неверные данные указывают на проблему в нативном коде. В таких случаях виртуальная машина Java завершается с ошибкой. Использование данной опции может приводить к снижению производительности.

`-Xfuture` - выполнение строгих проверок файлов классов. Для обратной совместимости с прошлыми версия JVM поддерживается нестрогий режим проверки.

`-Xnoclassgc` - отключает сбор мусора (garbage collection). Использование этой опции предотвратит очистку памяти загруженных классов, но повысит общее использование памяти, может приводить к ошибке OutOfMemoryError.

`-Xincgc` (только для версии 8) - включает инкрементальный сбор мусора (garbage collector).

Инкрементальный сбор мусора (garbage collector) по умолчанию отключён для снижения длительности пауз во время выполнения программы из-за конкурентной работы сборщика мусора. Инкрементальная сборка мусора требует значительной вычислительной мощности процессора.

`-Xloggc:file` - отображает каждое событий при сборке мусора (garbage collection) и выводит результаты в текстовый файл. Дополнительный ключ `-verbose:gc` добавляет к записям событий время (в секундах), которое прошло с момента первого события сборки мусора. Рекомендуется использовать локальное хранилище для того, чтобы избежать задержек при передаче информации по сети. Запись файл может быть прекращена при истечении свободного места, при этом запись будет продолжена в том же файле путём перезаписи прежних событий.

`-Xmnsize` или `-XX:NewSize` - задаёт размер для экстренного сборщика мусора (garbage collection).

`-Xms_n_` - указывается начальный размер пула выделяемой памяти. Данная величина должна быть кратна 1024 и выше, чем 1 Мб. Указание символа k или K определяет килобайты. Указание символа m или M - мегабайты.

`-Xrs` - отключает обработку сигналов ОС от Java VM (нажатие сочетаний клавиш выхода из консоли, закрытие окна консоли, завершение сеанса операционной системы или выключение устройства), используемую для реакции на события системы, например для своевременного сохранения дампа памяти на долговременное хранилище с целью последующего восстановления.

`-Xssn` - задаёт размер стека потока.

`-Xverify:mode` - задаёт режим валидации байткода. Валидация байткода проверяет, что файлы классов корректно сформированы и не нарушают заданных JVM ограничений.

Доступные режимы

`remote` - валидация классов из внешних источников (по умолчанию).

`all` - валидация всех классов.

`-XX:+AggressiveOpts` - включает режим экстремальной оптимизации.

Для получения информации по всем доступным опциям интерпретатора Java требуется использовать ключ

`-help` - перечень разрешённых опций интерпретатора Java.

javac

Компилятор языка программирования Java. Преобразует исходный код в промежуточный байткод, который хотя и не может быть выполнен непосредственно (как .exe файл, но может быть исполнен посредством интерпретатора java).

Существует два способа обработки файлов с исходными кодами:

1. При небольшом числе файлов они просто перечисляются в командной строке.
2. При значительном числе файлов их список, разделённый пробелами и новыми строками, ведётся во внешнем файле, который впоследствии указывается в команде после символа "@".

Файлы с исходными кодами должны иметь расширение `.java`. Файлы с классами должны иметь расширение `.class`, а также наименование, совпадающее с именем класса. Внутренние объявления классов при компиляции создают дополнительные файлы, наименование которых автоматически генерируется по имени исходного класса и внутреннего в формате: `[имя исходного класса]$[имя внутреннего класса].class`.

Следует размещать файлы с исходными кодами в дереве каталогов, отражающем их вложенность. Например, вы можете хранить все свои исходные коды в каталоге workspace, при этом исходный код класса `com.example.myClass` должен быть размещён в файле `workspace > com > example > myClass.java`.

По умолчанию компилятор помещает каждый файл класса в такую же директорию, в которой обнаружен файл с исходным кодом. Существует возможность указания каталога размещения файла при помощи команды `-d`.

Синтаксис

```
javac [ options ] [ sourcefiles ] [ classes ] [ @argfiles ]
```

Аргументы могут быть использованы в произвольном порядке.

`options` - параметры командной строки.

`sourcefiles` - один и более файлов с исходными кодами для компиляции.

`classes` - один и более классов для обработки аннотаций.

`@argfiles` - один и более файлов, содержащих списки опций и исходных файлов. Параметры JVM -J не допускаются в данном компоненте.

Опции компиляции можно изменять при помощи ключей компилятора javac:

`-version` - вывести версию компилятора

`-J` - свойство, передаваемое в JVM. Виртуальная машина может изменять своё поведение в зависимости от переданных параметров.

`-target` - указать версию JVM, для которой создаётся класс-файл.

`-source` - указать версию исходного кода.

`-bootclasspath` - указать путь, по которому можно найти классы, необходимые для запуска JVM.

`-cp` или `-classpath` - указание путей, по которым производится поиск классов, необходимых для запуска. Несколько путей разделяются между собой точкой с запятой (;) или двоеточием (:) в ОС Windows/Linux соответственно. Указанием `-classpath` перезаписывается переменная окружения `CLASSPATH`. Если не указаны переменная окружения `CLASSPATH` и параметр `-classpath`, то поиск класса осуществляется только в текущей директории.

`-Akey[=value]` - опции, передаваемые обработчикам аннотаций. Они не интерпретируются `javac` напрямую, а становятся доступными для частных обработчиков. Ключи должны быть отделены одной или более точкой ".".

`-Djava.ext.dirs=directories` - переопределение каталога размещения установленных расширений.

`-d` <каталог> - указание корневого каталога для файлов классов. Каталог должен существовать, т.к. javac не сможет создать его. Если класс входит в состав пакета, то javac сможет разместить файл класса в подкаталог в соответствии с именем пакета.

`-encoding` <кодировка> - выбор файла кодировки, например, EUC-JP или UTF-8. Если ключ `-encoding` не определён, то будет использован конвертер по умолчанию.

`-extdirs` <каталоги> - компиляция с использованием сторонних расширений другой версии платформы Java. Указываются каталоги размещения расширений через точку с запятой.

`-g` - Режим отладки, включая отображения локальных переменных. По умолчанию отображается только номер строки и исходный файл. Возможны следующие режимы:

- `g:none` - не выводить отладочную информацию
- `g:{keyword list}` - выводить указанные поля отладочной информации:
    - `source` - название файла с исходными кодами.
    - `lines` - номера строк с ошибками.
    - `vars` - значения локальных переменных.

`-nowarn` - отключение отображения предупреждений. Аналогичен ключу `-Xlint:none`.

`-s` <каталог> - определяет корневой каталог, в котором будут расположены файлы после компиляции. Файлы будут распределяться по подкаталогам на основании пространства имён пакета.

`-verbose` - включает текстовый вывод загруженных классов и наименований скомпилированных файлов.

`-deprecation` - отображает описание для каждого случая использования или переопределения устаревшего класса или метода. Без данного ключа javac отображает только статистику исходных файлов, в которых есть использование или переопределение устаревших классов или методов.

`-Werror` - завершение компиляции, при обнаружении предупреждений.

`-X` - расширенные параметры компиляции

`-Xcheck:jni` - дополнительные проверки для JNI кода.

`-Xstdout` - перенаправление вывода программы.

`-Xlint` - выводить предупреждения о некорректном коде программы. Доступны следующие варианты вывода предупреждений:

`-Xlint:all` - включает вывод всех типов рекомендуемых предупреждений. К типам предупреждений относятся внешне корректные синтаксические конструкции Java, которые могут обуславливать ошибки в коде. Например, ненужное приведение типов (тип cast), некорректное содержимое файла класса (тип classfile), использование устаревших методов (тип deprecation) и т.д.

`-Xlint:none` - отключает вывод предупреждений.

`-Xlint:name` - включает отображение указанного типа предупреждений.

`-Xlint:-name` - отключает отображение указанного типа предупреждений.

`-Xmaxerrs`/`-Xmaxwarns` - установить максимальное число выводимых ошибок/предупреждений.

`-Xbootclasspath/a` (`/p`) - заменить классы, необходимые для запуска компилятора. /a - поставить в начало последовательности классов. /p - поставить в конец последовательности классов.

`-Xprefer:{newer,source}` - определяет в каком файле осуществлять поиск типа: в файле исходного кода или в файле класса.

`-Xpkginfo:{always,legacy,nonempty}` - определяет обработку информационных файлов

пакетов

`-Xprint` - текстовое представление указанных типов (для отладки), для которых не будет осуществляться обработка аннотаций и компиляция.

`-XprintProcessorInfo` - вывод информации о запросах на обработку аннотаций.

`-XprintRounds` - вывод информации о начальном и последующих циклах обработки аннотаций.

Помимо стандартных опций доступно использование символов подстановки * и @ для перечисления объектов и ссылки на файл с перечнем объекта.

Для получения информации по всем доступным опциям компилятора требуется использовать ключ

`-help` - перечень разрешённых опций компилятора.

javadoc

Генератор документации к Java API. Позволяет создавать HTML-страницы из файлов с исходными кодами.

Синтаксис

```
javadoc [ options ] [ packagenames ] [ sourcefilenames ] [ -subpackages
pkg1:pkg2:… ] [ @argfiles ]
```

Следующие аргументы могут быть использованы в произвольном порядке.

`options` - аргументы модуля javadoc.

`packagenames` - дополнительные пакеты, разделённые пробелами, которые требуется включить в документацию (например, java.lang java.lang.reflect, java.awt). Символы автозамены не поддерживаются, для рекурсивного поиска пакетов требуется использовать ключ -subpackages. Компонент javadoc использует ключ -sourcepath для поиска пакетов.

`sourcefilenames` - список файлов с исходными кодами. Допускается использование символов автозамены, например '*'. Для исключения отдельных файлов следует воспользоваться символом "-". Для поиска файлов с исходными кодами ключ sourcepath не используется.

`subpackages pkg1:pkg2:…` - создаёт документацию из файлов с исходными кодами указанных пакетов и рекурсивно в их подпакетах. Используется в качестве альтернативы задания пакетам packagenames или файлам sourcefilenames.

`@argfiles` - один и более файлов, содержащих списки опций и исходных файлов. Параметры JVM

-J не допускаются в данном компоненте.

Параметры

`-overview path`filename` - указывает javadoc-файл, который должен идти в качестве обзорного раздела к документации. Данная опция доступна только при наличии двух и более пакетов. Заголовок документа указывается при помощи ключа `-doctitle`.

`-public` - выводит в документацию только public-классы.

`-protected` - выводит в документацию только protected- и public-классы. Режим по умолчанию.

`-package` - выводит в документацию пакеты, protected- и public-классы.

`-private` - выводит все классы.

`-doclet class` - указывает класс для запуска доклета - приложения, работающие со средством javadoc. Требуется указывать полное имя класса. Доклеты определяют содержание и формат вывода. Если опция не задана, то будет использоваться стандартный доклет, который создаёт HTML-страницу.

`-docletpath classpathlist` - указывает путь к файлу класса для запуска доклета (указывается в опции -doclet)

`-sourcepath sourcepathlist` - определяет пути поиска файлов исходного кода с расширением .java при поиске имён пакетов или указанных в команде -subpackages. Для ввода нескольких путей следует разделять их точкой с запятой. javadoc осуществляет поиск в каталогах и всех подкаталогах.

`-classpath classpathlist` - указывает пути, по которым javadoc будет искать файлы классов с расширением .class files.

`-subpackages package1:package2:…` - создаёт документацию из файлов с исходными кодами, в указанных пакетах и рекурсивно найденных подпакетов. Данная опция полезна при добавлении новых исходников в подпакеты. Например:

```
javadoc -d docs -sourcepath C:\user\src -subpackages \ java:javax.swing
```

Команда из примера выше создаёт документы для пакетов java и javax.swing, а также для всех их подпакетов. Опцию `-subpackages` можно использовать для исключения из списка пакетов для генерации документов (совместно с ключом `exclude`).

`-exclude packagename1:packagename2:…` - исключает пакеты и их подпакеты из списка на генерацию документации. Список формируется при помощи команды -subpackages.

`-bootclasspath classpathlist` - указывает пути размещения загрузочных классов. Каждый путь отделяется точкой с запятой.

`-extdirs dirlist` - указывает каталоги размещения расширений. Каждый путь отделяется точкой с запятой.

`-verbose` - расширенный текстовый вывод при запуске команды javadoc.

`-quiet` - отключение информационных сообщений.

`-locale` language_country_variant - указывает региональные настройки и язык документации. В качестве аргумента используется определённые в `java.util.Locale` константы. Например: en_US (English, United States) или en_US_WIN (Windows variant). Примечание: если используется опция -locale, то она должна быть размещена самой первой (слева), в противном случае будут ошибки.

`-Jflag` - флаги управления JVM.

`-d` directory - указывает директорию назначения, куда будут размещаться созданные файлы.

`-version` - включает тег @version в документы.

`-author` - включает тег @author в документы.

`-charset name` - указывает кодировку для генерируемого документа.

Для получения информации по всем доступным опциям генератора документации требуется использовать ключ.

`-help` - перечень разрешённых опций генератора документации.

javap

Дизассемблер для файлов классов Java, который разбирает класс-файл. Выводимая информация варьируется в зависимости от используемых опций. По умолчанию javap выводит название пакета, а также protected и public поля и методы анализируемого класса.

Синтаксис

```
javap [ options ] classes
```

Параметры

`options` - параметры командной строки.

`classes` - список из одного или более классов, разделённых пробелами.

`-l` - вывод списка локальных переменных.

`-public` - выводит в документацию только public-классы.

`-protected` - выводит в документацию только protected- и public-классы.

`-package` - выводит в документацию пакеты, protected- и public-классы.

`-private` - выводит все классы. Режим по умолчанию.

`-[J]_flag_` - указание параметров JVM.

`-s` - показывает пакет, в котором расположен класс, а также его -protected и -public поля и методы.

`-c` - вывод дизассмеблированного кода, т.е. инструкций, включённых в байткод.

`-verbose` - выводит размер стека, число локальных переменных и аргументов для метода.

`-help` - выводит информацию об опциях использования.

jdeps

jdeps - анализатор зависимостей классов Java.

Команда jdeps отображает зависимости файлов классов Java на уровне пакета или класса. Входным классом может быть путь к файлу `.class`, каталогу, JAR-файлу или полное имя класса для анализа всех файлов класса. Параметры определяют вывод данных. По умолчанию команда jdeps записывает зависимости в системный вывод.

Синтаксис

```
jdeps [options] path
```

`options` - параметры командной строки.

`path` - Путь к файлу `.class`, каталогу или JAR-файлу для анализа.

Общие параметры

`-dotoutput dir` или `--dot-output dir` - путь к каталогу для вывода DOT-файлов. Если указан этот параметр, команда jdeps генерирует один .dot-файл для каждого анализируемого архива с именем archive-file-name.dot, в котором перечислены зависимости, а также сводный файл с именем summary.dot, в котором перечислены зависимости между файлами архива.

`-s` или `-summary` - выводит только сводку зависимостей.

`-v` или `-verbose` - выводит все зависимости на уровне классов. Это эквивалентно команде `-verbose:class -filter:none`.

`-verbose:package` - выводит зависимости на уровне пакета, исключая по умолчанию зависимости внутри того же пакета.

`-verbose:class` - выводит зависимости на уровне класса, исключая по умолчанию зависимости внутри того же архива.

`-apionly` или `--api-only` - ограничивает анализ API, например, зависимостями из сигнатуры открытых и защищенных членов открытых классов, включая тип поля, типы параметров методов, возвращаемый тип и типы проверяемых исключений.

`-jdkinternals` или `--jdk-internals` - находит зависимости на уровне класса во внутренних API JDK. По умолчанию этот параметр анализирует все классы, указанные в параметре `--classpath`, и входные файлы, если не указан параметр `-include`. Этот параметр нельзя использовать с параметрами `-p`, `-e` и `-s`.

> **Важно:**
>
> Внутренние API JDK недоступны.

`-cp path`, `-classpath path` или `--class-path path` - указывает, где искать файлы классов.

`--module-path module-path` - указывает путь к модулю.

`--upgrade-module-path module-path` - указывает путь к модулю обновления.

`--system java-home` - указывает альтернативный путь к системным модулям.

`--add-modules module-name[, module-name…]` - Добавляет модули в корневой набор для анализа.

`--multi-release version` - Указывает версию при обработке JAR-файлов с несколькими релизами. version должна быть целым числом >=9 или base.

`-q` или `-quiet` - не отображает отсутствующие зависимости из вывода `-generate-module-info`.

`-version` или `--version` - выводит информацию о версии.

Параметры анализа зависимостей модулей

`-m module-name` или `--module module-name` - указывает корневой модуль для анализа.

`--generate-module-info dir` - генерирует файл module-info.java в указанном каталоге. Будут проанализированы указанные JAR-файлы. Этот параметр нельзя использовать с параметрами `--dot-output` или `--class-path`. Используйте параметр `--generate-open-module` для открытых модулей.

`--generate-open-module dir` - генерирует файл module-info.java для указанных JAR-файлов в указанном каталоге в качестве открытых модулей. Этот параметр нельзя использовать с параметрами `--dot-output` или `--class-path`.

`--check module-name [, module-name…]` - анализирует зависимости указанных модулей. Выводит дескриптор модуля, результирующие зависимости модулей после анализа и граф после сокращения переходов. Также выявляет любые неиспользуемые квалифицированные экспорты.

`--list-deps` - выводит список зависимостей модулей, а также имена пакетов внутренних API JDK (если они указаны). Этот параметр транзитивно анализирует библиотеки в пути классов и пути модулей, если они указаны. Используйте параметр `--no-recursive` для анализа нетранзитивных зависимостей.

`--list-reduced-deps` - то же, что и `--list-deps`, но без вывода подразумеваемых ребер чтения из графа модулей. Если модуль M1 читает M2, и M2 требует транзитивного чтения M3, то чтение M3 модулем M1 подразумевается и не отображается в графе.

`--print-module-deps` - то же, что и `--list-reduced-deps`, но с выводом списка зависимостей модулей, разделенных запятыми. Вывод можно использовать с помощью `jlink --add-modules` для создания пользовательского образа, содержащего эти модули и их транзитивные зависимости. См. описание инструмента jlink.

`--ignore-missing-deps` - Игнорировать отсутствующие зависимости.

Параметры фильтрации зависимостей

`-p pkg_name`, `-package pkg_name` или `--package pkg_name` - находит зависимости, соответствующие указанному имени пакета. Этот параметр можно указывать несколько раз для разных пакетов. Параметры `-p` и `-e` являются взаимоисключающими.

`-e regex`, `-regex regex` или `--regex regex` - находит зависимости, соответствующие

указанному шаблону. Параметры `-p` и `-e` являются взаимоисключающими.

`--require module-name` - находит зависимости, соответствующие заданному имени модуля (может быть указано несколько раз). Параметры `--package`, `--regex` и `--require` являются взаимоисключающими.

`-f regex` или `-filter regex` - фильтрует зависимости, соответствующие заданному шаблону. Если указать несколько раз, будет выбран последний.

`-filter:package` - фильтрует зависимости внутри одного пакета. Это значение по умолчанию.

`-filter:archive` - фильтрует зависимости внутри одного архива.

`-filter:module` - фильтрует зависимости внутри одного модуля.

`-filter:none` - фильтрация с помощью `-filter:package` и `-filter:archive` не применяется. Фильтрация, указанная с помощью параметра `-filter`, по-прежнему применяется.

`--missing-deps` - находит отсутствующие зависимости. Этот параметр нельзя использовать с параметрами `-p`, `-e` и `-s`.

Параметры фильтрации анализируемых классов

`-include regex` - ограничивает анализ классами, соответствующими шаблону. Этот параметр фильтрует список анализируемых классов. Его можно использовать вместе с `-p` и `-e`, которые применяют шаблон к зависимостям.

`-P` или `-profile` - Отображает профиль, содержащий пакет.

`-R` или `--recursive` - Рекурсивно обходит все зависимости. Параметр `-R` подразумевает `-filter:none`. Если указаны параметры `-p`, `-e` или `-f`, анализируются только соответствующие зависимости.

`--no-recursive` - Не использовать рекурсию при обходе зависимостей.

`-I` или `--inverse` - анализирует зависимости в соответствии с другими заданными параметрами, а затем находит все артефакты, которые прямо и косвенно зависят от соответствующих узлов. Это эквивалентно обратному анализу `--compile-time` и выводу сводки зависимостей. Этот параметр необходимо использовать с параметрами `--require`, `--package` или `--regex`.

`--compile-time` - анализирует транзитивные зависимости на этапе компиляции, например, на этапе компиляции, указанном в параметре `-R`. Анализирует зависимости в соответствии с другими указанными параметрами. Если зависимость обнаружена в каталоге, JAR-файле или модуле, анализируются все классы в соответствующем архиве.

Примеры анализа зависимостей

Пример получения зависимостей для jar файла:

```
$ jdeps AwtDemo.jar
AwtDemo.jar -> java.base
AwtDemo.jar -> java.desktop
   <unnamed>                                          -> java.awt
java.desktop
   <unnamed>                                          -> java.awt.event
java.desktop
   <unnamed>                                          -> java.lang
java.base
```

Пример получения зависимостей в виде списка модулей для использования с jlink:

```
$ jdeps --print-module-deps AwtDemo.jar
java.base,java.desktop
```

Пример получения списка всех модулей, которые зависят от `java.xml`:

```
$ jdeps --require  java.xml --inverse

Inverse transitive dependences on [java.xml]
java.xml <- java.se
java.xml <- jdk.xml.dom
java.xml <- java.desktop <- java.se
java.xml <- java.desktop <- jdk.accessibility
java.xml <- java.desktop <- jdk.editpad
java.xml <- java.desktop <- jdk.hotspot.agent
java.xml <- java.desktop <- jdk.jconsole
java.xml <- java.desktop <- jdk.jpackage
java.xml <- java.desktop <- jdk.unsupported.desktop
java.xml <- java.prefs <- java.desktop
java.xml <- java.prefs <- java.se
java.xml <- java.prefs <- jdk.jshell
java.xml <- java.sql <- java.se
java.xml <- java.xml.crypto <- java.se
java.xml <- java.sql <- java.sql.rowset <- java.se
```

jlink

jlink - инструмент сборки и оптимизации набора модулей с их зависимостями в индивидуальный runtime-образ, доступен в версии 11 и выше.

Синтаксис

```
jlink [options] --module-path modulepath --add-modules module[,module…]
```

`options` - параметры командной строки. Разделяются пробелами.

`modulepath` - путь, по которому jlink ищет доступные модули. Эти модули могут быть модульными JAR-файлами, JMOD-файлами, или развернутыми модулями.

`module` - имя модуля добавляемого в runtime-образ. jlink добавит этот модуль со всеми зависимостями.

> **Примечание:**
>
> Ответственность за обновление нестандартных образов рантайма собранных jlink лежит на разработчике этих образов.

Параметры

`--add-modules mod [,mod…]` - добавляет именованные модули mod в корневой набор модулей по-умолчанию. По-умолчанию корневой набор модулей является пустым.

`--bind-services` - линкует модули сервис провайдера и их зависимости.

`--compress` <compress> - включает компрессию:

- Версии JDK 11 и 17

  `--compress={0|1|2}`

  - 0 - Без компрессии
  - 1 - Разделяемые константы
  - 2 - ZIP

- Версия JDK 21

`--compress=zip-[0-9]` - где `zip-0` не обеспечивает сжатия, а `zip-9` обеспечивает наилучшее сжатие.

`--disable-plugin pluginname` - отключает заданные плагины.

`--endian {little|big}` - определяет байтовый порядок в сгенерированном образе. Порядком по умолчанию является порядок в системе пользователя.

`-h` или `--help` - печать сообщения помощи

`--ignore-signing-information` - не выдавать критическую ошибку если подписанные модульные JAR-файлы линкуются в runtime-образ. Относящиеся к подписи файлы из состава подписанных модульных JAR-файлов не будут скопированы в runtime-образ.

`--launcher command=module` или `--launcher command=module/main` - задает имя команды запуска для модуля или имя команды для модуля и основного класса (имена модуля и основного класса разделяются косой чертой /).

`--limit-modules mod [,mod…]` - ограничивает совокупность наблюдаемых модулей модулями в транзитивном замыкании названных модулей, mod, плюс основной модуль, если таковой имеется, плюс любые дополнительные модули, указанные в `--add-modules` опции.

`--list-plugins` - перечисляет доступные плагины, к которым вы можете получить доступ через параметры командной строки; см. плагины jlink.

`-p` или `--module-path modulepath` - задает путь к модулю. Если этот параметр не указан, то путь к модулю по умолчанию - $JAVA_HOME/jmods. Этот каталог содержит модуль java.base и другие стандартные модули и модули JDK. Если этот параметр указан, но модуль java.base не может быть разрешен из него, тогда команда jlink добавляет $JAVA_HOME/jmods к пути до модуля.

`--no-header-files` - исключает файлы заголовков.

`--no-man-pages` - исключает страницы руководства.

`--output path` - задает расположение сгенерированного runtime-образа.

`--save-opts filename` - сохраняет параметры jlink в указанном файле.

`--suggest-providers [name, …]` - предлагать поставщиков, реализующих указанные типы услуг, из пути к модулю.

`--version` - печатает информацию о версии.

`@filename` - считывает параметры из указанного файла. Файл параметров - это текстовый файл, содержащий параметры и значения, которые вы обычно вводите в командной строке. Параметры могут отображаться в одной строке или в нескольких строках. Вы не можете указывать

переменные среды для имен путей. Вы можете закомментировать строки, добавив символ решетки # в начало строки.

Примеры

В данном примере производится сборка образа JDK с модулями `java.desktop` и `java.xml`, без документации (man) и заголовочных файлов. В качестве JDK для сборки используется Axiom JDK Pro 17.0.17, в других версиях консольный вывод может отличаться.

```
export JAVA_HOME=<Путь к AxiomJDK>
export OUTPUT_DIRECTORY=<Путь к директории куда будет записан собранный JDK>
$JAVA_HOME/bin/jlink --add-modules java.desktop,java.xml --output
$OUTPUT_DIRECTORY --no-man-pages --no-header-files
```

Где `<Путь к AxiomJDK>` это путь к Axiom JDK Pro установленной на машине, на которой производится сборка.

В сгенерированном образе будут созданы следующие файлы и каталоги:

```
ls $OUTPUT_DIRECTORY
```

Вывод:

```
bin  conf  legal  lib  release
```

В сгенерированный образ будут добавлены запрашиваемые модули и их зависимости:

```
$OUTPUT_DIRECTORY/bin/java --list-modules
```

Вывод:

```
java.base@17.0.17
java.datatransfer@17.0.17
java.desktop@17.0.17
java.prefs@17.0.17
java.xml@17.0.17
```

В следующем примере производится сборка JRE со следующими утилитами: `jcmd`, `jconsole`, `jstack`, `jstat`, `jstatd`.

```
export JAVA_HOME=<Путь к AxiomJDK>
export OUTPUT_DIRECTORY=<Путь к директории куда будет записан собранный JDK>
```

```
$JAVA_HOME/bin/jlink --add-modules jdk.jcmd,jdk.jstatd,jdk.jconsole --output
$OUTPUT_DIRECTORY
```

В сгенерированном образе будут созданы следующие файлы и каталоги:

```
ls $OUTPUT_DIRECTORY
```

Вывод:

```
bin  conf  include  legal  lib  man  release
```

В каталоге `bin` созданного образа будут находиться инструменты соответствующие модулям:

```
ls $OUTPUT_DIRECTORY/bin
```

Вывод:

```
java  jcmd  jconsole  jinfo  jmap  jps  jstack  jstat  jstatd  keytool
rmiregistry
```

Список добавленных в образ модулей и их зависимостей:

```
java.base@17.0.17
java.datatransfer@17.0.17
java.desktop@17.0.17
java.logging@17.0.17
java.management@17.0.17
java.management.rmi@17.0.17
java.naming@17.0.17
java.prefs@17.0.17
java.rmi@17.0.17
java.security.sasl@17.0.17
java.xml@17.0.17
jdk.attach@17.0.17
jdk.internal.jvmstat@17.0.17
jdk.jcmd@17.0.17
jdk.jconsole@17.0.17
jdk.jstatd@17.0.17
jdk.management@17.0.17
jdk.management.agent@17.0.1
```

jdb

Отладчик Java jdb требуется для обнаружения и исправления ошибок в программах Java.

Синтаксис

```
jdb [ options ] [ class ] [ arguments ]
```

`options` - параметры командной строки.

`class` - имя класса для отладки.

`arguments` - аргументы передаваемые в метод main() класса.

Параметры

Большинство опций интерпретатора java доступно и в jdb, например, `-D`, `-classpath` или `-X<_option_>`.

`-sourcepath <dir1:dir2:…\>` - использует указанные пути для поиска исходных файлов. Если опция не задана, поиск осуществляется в текущем каталоге.

`-attach` - связывает отладчик с предварительно запущенной VM при помощи механизма по умолчанию.

`-listen` - ожидает момента подключения запущенной VM к указанному адресу.

`-listenany` - ожидает момента подключения запущенной VM к любому (any) адресу.

`-launch` - запускает отладку при старте jdb. Данная опция позволяет избежать необходимости ручного выполнения команды run.

`-listconnectors` - вывод списка подключений данной VM.

`-connect <connector-name\>:<name1\>=<value1\>,…` - подключается к целевой VM по её имени с перечисленными значениями аргументов.

`-dbgtrace [flags]` - вывод отладочной информации jdb.

`-tclient` - запуск приложения в Java HotSpot(tm) VM (Client).

`-tserver` - запуск приложения в Java HotSpot(tm) VM (Server).

`-[J]_option_` - передаёт опции загрузчика Java-приложений виртуальной машине Java.

Для получения информации по всем доступным опциям отладчика Java требуется использовать ключ

`-help` - перечень разрешённых опций отладчика Java.

5. Действия после сбоев и ошибок эксплуатации

Все ошибки выводятся в виде исключений (Throwable/Exception) либо в стандартный поток вывода (по умолчанию), либо в соответствии с конфигурацией, заданной пользователем. В случае аварийного завершения виртуальной машины java будет создан аварийный дамп памяти (если иное не установлено в конфигурации среды функционирования ОО - операционной системы). Для исправления ошибок пользователю рекомендуется ознакомится с описанием исключения и внести необходимые корректировки.

AXIOM JDK

Axiom JDK Pro
Руководство пользователя