УДК 004.632

СИСТЕМА БЕЗОПАСНОГО ВЫПОЛНЕНИЯ ДИНАМИЧЕСКИ ГЕНЕРИРУЕМОГО JAVA-КОДА

В.А. МАКАРЫЧЕВА, М.Ю. МАКАРЫЧЕВ
(Представлено: канд. техн. наук, доц. А.Ф. ОСЬКИН)

Представлен и подробно рассмотрен способ компиляции и безопасного исполнения исходного Java-кода во время работы программы.

Введение. В процессе разработки программного обеспечения иногда возникают проблемы, связанные с неопределённостью структуры программы на момент её компиляции. Другими словами, программа в некоторых местах имеет доступ к исходному коду, который требуется выполнить, только во время её работы. Технология JSP (Java Server Pages) является примером динамической генерации, компиляции и выполнения Java-кода. Транслятор JSP преобразует .jsp файлы в Java-сервлеты (файлы исходного кода), которые компилируются в момент выполнения и загружаются контейнером сервлетов.

Основная часть. Для решения задач этого класса Java предоставляет пакет javax.tools, который имеет следующие преимущества:

– Входит в состав JDK, представляет собой стандартный API, разработанный в рамках Java Community Process (JSR 199).

– Используется исходный код Java, а не байт-код. Это позволяет избежать запуска кода, который имеет ошибки компиляции, и предотвращает от некоторых уязвимостей JVM.

– Используется один механизм для генерации и загрузки кода, не ограничивая программиста использованием исходного кода на базе файлов.

– Используется проверенная версия компилятора.

– Поддерживается переносимость между реализациями JDK версии 1.6 и выше от любых производителей.

– В отличие от систем, основанных на интерпретации, загруженные классы могут пользоваться всеми оптимизациями, выполняемыми JRE во время выполнения [1].

Пакет javax.tools предоставляет настолько общие классы и интерфейсы, что позволяет не только загружать исходный код из файлов, но и получать его из различных объектов-источников.

Компиляция исходного кода требует следующих компонентов:

– Объект classpath, через который компилятор находит классы библиотек. Он состоит из списка каталогов файловой системы и архивных файлов (JAR или ZIP), которые содержать скомпилированные .class-файлы.

– javac options – параметры компилятора javac.

– Source files – один или несколько исходных .java-файлов. Объект JavaFileManager предоставляет абстрактную файловую систему, которая привязывает имена исходных и получившихся в результате файлов к экземплярам объектов JavaFileObject. В данном случае файл означает связь между уникальным именем и последовательностью байтов. Клиенту не обязательно использовать настоящую файловую систему. Например, JavaFileManager может управлять привязками между именами классов и объектами CharSequence, которые содержат исходный Java-код для компиляции.

– Output directories – каталоги, в которые компилятор записывает .class-файлы, содержащие байт-код.

– compiler (компилятор). Класс JavaCompiler создаёт объекты JavaCompiler.CompilationTask, которые компилируют исходный код объектов JavaFileObject SOURCE в объекте JavaFileManager, создавая новые выходные JavaFileObject CLASS файлы и объекты типа Diagnostic (предупреждения и ошибки). Статический метод ToolProvider.getSystemJavaCompiler() возвращает экземпляр компилятора.

– Compiler warnings and errors – сообщения от компилятора с предупреждениями и ошибками, которые реализуются классами Diagnostic и DiagnosticListener.

Далее подробнее рассматривается компиляция исходного кода, который содержится в объектах CharSequence (ими могут быть String, StringBuffer и StringBuilder).

Чтобы скомпилировать код из объекта (или объектов) CharSequence, необходимо сконструировать свой класс, к примеру, CharSequenceJavaCompiler, который будет иметь следующий API:

– Конструктор с параметрами ClassLoader и Iterable<String>. Первый параметр представляет собой загрузчик классов, который необходимо передать компилятору, чтобы тот мог находить зависимые классы. Второй параметр представляет собой список опций компилятора.

– Метод компиляции compile с параметрами String и CharSequence. Первый параметр – имя класса, второй – исходный код этого класса. Если требуется скомпилировать сразу несколько классов в разных объектах CharSequence, этот метод должен быть перегружен. В этом случае он может принимать параметр Map<String, CharSequence>, ключи которого – имена классов, а значения – исходный код. Метод должен возвратить объект типа Map<String, Class<T>>, ключи которого – имена классов, а значения – объекты типа Class, которые были загружены. Также этот метод должен заполнить список объектов типа Diagnostic.

– Метод getClassLoader, который возвращает загрузчик классов, созданный компилятором во время генерации .class-файлов, чтобы из него можно было загружать другие классы или ресурсы.

– Метод loadClass, который принимает String – имя класса и возвращает объект Class. Метод compile, используя метод loadClass, может загрузить несколько классов, включая вложенные, с одного CharSequence объекта.

Для работы всех методов класса CharSequenceJavaCompiler необходимо реализовать интерфейсы JavaFileObject и JavaFileManager. Реализация первого интерфейса JavaFileObjectImpl будет хранить объекты CharSequence с исходным кодом и выходные CLASS-объекты, которые создаст компилятор. Ключевой метод этого класса должен предоставлять текст исходного кода. Реализация второго интерфейса JavaFileManagerImpl связывает имена с экземплярами JavaFileObjectImpl для управления последовательностями исходного кода и байт-кода, созданного компилятором. На самом деле JavaFileManagerImpl может не реализовывать интерфейс напрямую, вместо этого он может расширить класс, то есть стать наследником ForwardingJavaFileManager<JavaFileManager>.

Теперь с этими вспомогательными классами можно определить класс CharSequenceJavaCompiler. Он создается с объектом ClassLoader (загрузчиком классов времени исполнения) и параметрами компилятора. В нем используется метод ToolProvider.getSystemJavaCompiler() для получения экземпляра JavaCompiler, затем создается JavaFileManagerImpl, который перенаправляется в стандартный менеджер файлов компилятора.

Метод compile() проходит по поданной на вход таблице ключ-значение, создавая объекты JavaFileObjectImpl из каждой пары имя/объект CharSequence и добавляя их в JavaFileManager, чтобы JavaCompiler нашел их при вызове метода getFileForInput() менеджера файлов. Метод compile() затем создает объект JavaCompiler.Task и запускает его. Сбои приводят к выдаче исключительной ситуации CharSequenceJavaCompilerException. Далее для каждого элемента исходного кода, переданного в метод compile(), загружается полученный объект Class и помещается в выходную коллекцию типа Map.

Загрузчик классов, связанный с CharSequenceJavaCompiler – это объект типа ClassLoaderImpl, который ищет байт-код для класса в объекте типа JavaFileManagerImpl, возвращая .class файлы, созданные компилятором [2].

Заключение. Использование javax.tools API полностью является безопасным. Например, класс JavaFileManager может отказать в записи любого непредвиденного .class-файла, бросая SecurityException. Также JavaFileManager может допускать только сгенерированные имена классов и пакетов, которые пользователь не сможет предугадать или подменить. Кроме этого рекомендуется использовать специальные объекты SecurityManager или ClassLoader, предотвращающие небезопасное поведение, загрузку анонимных классов или других классов, которые не контролируются непосредственно разработчиком.

ЛИТЕРАТУРА

1. Oracle [Электронный ресурс]. – Package javax.tools. – Режим доступа: https://www.ibm.com/developerworks/ru/library/j-jcomp/index.html. – Дата доступа: 20.09.18.
2. IBM [Электронный ресурс] // Создание динамических приложений с помощью javax.tools. – Режим доступа: https://www.ibm.com/developerworks/ru/library/j-jcomp/index.html. – Дата доступа: 20.09.18.