

# Java:

## *торжественное обращение с jar и атрибутами MANIFEST.MF*

**Алексей Саенко**

Java-приложения гораздо удобнее собирать в один jar-файл, нежели хранить сложную структуру нескольких тысяч файлов. Однако плюсы использования jar-файлов этим удобством не ограничиваются, иногда может возникнуть необходимость передавать параметры из манифеста.

Многие Java-приложения поставляются в виде одного или нескольких jar-файлов (Java ARchive), которые по сути являются обыкновенными zip-архивами. В них обычно, кроме байт-кода программы, размещаются дополнительные ресурсы и конфигурационные файлы.

Такой подход позволяет:

- подписывать содержимое jar-файла, повышая таким образом уровень безопасности;
- сокращать время загрузки приложений за счет отсутствия необходимости открытия нового соединения для каждого файла;

- сокращать объем приложения;
- упростить процесс создания библиотеки;
- осуществлять контроль версий.

Полноценно работать с технологией JAR можно как с помощью утилиты jar, входящей в состав JDK (Java Development Kit), так и с использованием классов JAR API.

Первый способ распространен повсеместно, в то время как второй, зачастую незаслуженно, обделен вниманием и, как показывает практика, напрасно – во многих случаях это может существенно облегчить жизнь раз-

работчика. Именно поэтому в данной статье будут описаны несколько способов применения JAR API для вызова произвольных методов из jar-файла, записи и чтения атрибутов манифеста. (Манифест – (устар.) торжественное письменное обращение верховной власти к народу в связи с важным политическим событием, торжественной датой и т. д. (Толковый словарь русского языка).

### Утилита jar

Прежде всего следует уделить некоторое внимание утилите jar (<http://java.sun.com/javase/6/docs/technotes/tools/>

solaris/jar.html). Это консольное приложение, запускаемое с набором параметров.

Примеры вызова утилиты:

```
// Создание нового jar-файла
jar cf file.jar список_файлов

// Просмотр содержимого архива
jar tf file.jar

// Извлечение содержимого из jar-файла
jar xf file.jar
```

## MANIFEST.MF

JAR File Specification (<http://java.sun.com/javase/6/docs/technotes/guides/jar/jar.html>), позволяет расположить в архиве file.jar только один манифест с дополнительной служебной информацией META-INF/MANIFEST.MF, представляющий собой текстовый файл в кодировке UTF-8. Если он не был задан при создании jar-файла, используется манифест, который по умолчанию содержит информацию о своей версии и JDK, в которой был создан конкретный jar-файл:

```
Manifest-Version: 1.0
Created-By: 1.6.0 (Sun Microsystems Inc.)
```

Условно содержимое можно разделить на главную группу (содержит определенные стандарты атрибуты, некоторые из них приведены в **таблице 1**) и индивидуальную группу (состав которой определяется произвольно).

Имена произвольных атрибутов могут содержать только цифры, строчные и заглавные буквы латинского алфавита и знак «\_», а по длине существует ограничение – 70 символов. К значению атрибута никаких требований, кроме элементарной логики, не предъявляется.

Более подробно структуру и состав JAR и META-INF/MANIFEST.MF в рамках этого текста мы затрагивать не будем по той простой причине, что любой желающий может прочесть более подробно спецификацию JAR.

## Использование JAR API

В последней на текущий момент Java 1.6 к JAR API относятся классы пакета java.util.jar, а также классы java.net.JarURLConnection и java.net.URLClassLoader. Ограничимся рассмотрением наиболее используемых классов (см. **таблицу 2**).

### Запуск методов класса

Прежде всего рассмотрим пример, который будет запускать метод public static void main(String[] args) из произвольного класса в jar-файле. Предположим, что у нас есть некий файл file.jar, который содержит класс pkg.Main:

```
package pkg;

public class Main {
```

Таблица 1. Атрибуты главной группы манифеста

Атрибут	Описание
Manifest-Version	Номер версии файла-манифеста. Определяется спецификацией как регулярное выражение следующего вида: цифра+{цифра+}*
Created-By	Версия и поставщик платформы Java, с помощью которой был создан манифест. Автоматически генерируется утилитой jar
Signature-Version	Версия подписи jar-файла. Определяется так же, как и Manifest-Version
Class-Path	Относительные указатели ресурсов (URL) для всех используемых дополнительных классов и библиотек
Main-Class	Относительный путь к главному классу приложения, который должен содержать точку входа – метод main(String[] args). Значение атрибута не должно содержать расширение .class. Атрибут определяется для автономных настольных приложений, которые собраны в выполняемые jar-файлы, которые могут быть запущены виртуальной машиной Java напрямую командой: java -jar file.jar

Таблица 2. Наиболее используемые классы в JAR API

Полное название класса	Описание
java.util.jar.Attributes	Таблица соответствий между именем атрибута и его значением
java.util.jar.Attributes.Name	Перечисление в виде констант всех имен атрибутов главной группы
java.util.jar.JarEntry	Класс описывает элемент jar-файла, например, файл с расширением .class
java.util.jar.JarFile	Наследник класса java.util.zip.ZipFile с поддержкой манифеста. Используется для чтения содержимого jar-файла
java.util.jar.JarInputStream	Наследник класса java.util.zip.ZipInputStream с поддержкой манифеста. Используется для чтения содержимого jar-файла из любого входящего потока
java.util.jar.JarOutputStream	Наследник класса java.util.zip.ZipOutputStream с поддержкой записи манифеста. Используется для записи содержимого jar-файла в любой исходящий поток
java.util.jar.Manifest	Класс используется для работы с манифестом и его атрибутами
java.net.JarURLConnection	Соединение с jar-файлом или его элементом с помощью указателя ресурсов вида: ar:<url_к_jar_файлу>!/(<элемент_jar_файла>), например: jar:http://www.site.org/folder/file.jar!/org/site/Clazz.class
java.net.URLClassLoader	Используется для загрузки классов и ресурсов из classpath (может определяться, к примеру, атрибутом манифеста Class-Path)

```
public static void main(String[] args) {
    System.out.println("main(): запущен");
}
```

MANIFEST.MF архива имеет следующий вид:

```
Manifest-Version: 1.0
Created-By: 1.6.0_03-b05 (Sun Microsystems Inc.)
Main-Class: pkg.Main
```

Вначале необходимо получить URL по имени файла:

```
URL fileUrl = new File("file.jar").toURL();
URL url = new URL("jar", "", fileUrl + "!/");
```

Сам вызов осуществляется следующим образом:

```
public static final String MAIN_METHOD = "main";
public void runMainMethod(String className) throws
    Exception {
    String[] args = new String[1];
    Class clazz;
    Method mainMethod;
    // Создадим ClassLoader
    URLClassLoader urlClassLoader =
        new URLClassLoader(new URL[] { url });
    // Загрузка класса класслоадером
    clazz = urlClassLoader.loadClass(className);
    // Получение main-метода и проверка, является ли он
    // точкой входа
    mainMethod = clazz.getMethod(MAIN_METHOD,
        args.getClass());
    mainMethod.setAccessible(true);
    int mods = mainMethod.getModifiers();
    if (mainMethod.getReturnType() != void.class
        || !Modifier.isStatic(mods) ||
        !Modifier.isPublic(mods)) {
        throw new NoSuchMethodException(MAIN_METHOD);
    }
}
```

```

    } else {
        // Запуск метода с объектом, у которого
        // надо выполнить метод, и параметрами.
        // null показывает, что метод статический
        mainMethod.invoke(null, args);
    }
}

```

Теперь попробуем задействовать JAR API для решения несколько усложненной задачи – теперь необходимо запустить метод `main()` главного класса `jar`-файла. Для этого нам надо лишь вызвать уже имеющийся метод `runMainMethod()` с именем требуемого класса, которое можно получить из манифеста по имени атрибута `Main-Class`:

```

private String getMainClassName() throws IOException {
    JarURLConnection connection = ␣
        (JarURLConnection) url.openConnection();
    Attributes attributes = ␣
        connection.getMainAttributes();
    return attributes.getValue(Attributes.Name.MAIN_CLASS);
}

```

После вызова метода `runMainMethod()` в консоли появится надпись:

```
main(): запущен
```

## Генерация манифеста с атрибутами

После того как мы успешно запустили метод `main()` главного класса `jar`-файла, попробуем поработать с атрибутами манифеста. Начнем с создания манифеста, а затем попробуем прочитать записанные атрибуты. Итак, класс `JarAttributeWriter` будет генерировать все атрибуты:

```

import java.io.ByteArrayInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.jar.Manifest;

public class JarAttributeWriter {
    private static final String LINE_TEMPLATE = "%s: %s\n";

    // Метод генерирует текстовый файл - манифест с именем
    // filename
    public void generateManifest(String filename) {
        StringBuffer buf = new StringBuffer();
        buf.append(getLine("Manifest-Version", "1.0"));
        buf.append(getLine("Created-By", ␣
            this.getClass().getName()));
        buf.append(getLine("Attribute_1", "Value_1"));
        buf.append(getLine("Attribute_2", "Value_2"));
        try {
            InputStream inputStream = ␣
                new ByteArrayInputStream ␣
                    (buf.toString().getBytes("UTF-8"));

            Manifest manifest = ␣
                new Manifest(inputStream);
            OutputStream outputStream = ␣
                new FileOutputStream(filename);
            manifest.write(outputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Метод возвращает одну строку манифеста. Метод format() -
    // аналог printf в C - осуществляет форматированный вывод
    private String getLine(String attributeName, ␣
        String attributeValue) {
        return String.format(LINE_TEMPLATE, attributeName, ␣
            attributeValue);
    }
}

```

В результате файл манифеста будет содержать:

```

Manifest-Version: 1.0
Created-By: JarAttributeWriter
Attribute_1: Value_1
Attribute_2: Value_2

```

## Чтение атрибутов из манифеста

Получившийся файл можно использовать в качестве манифеста для `jar`-архива – это довольно просто, поэтому предположим, что этот манифест уже лежит в `jar`-файле и требуется считать из него все атрибуты или только некоторые. Эту работу будет выполнять класс `JarAttributeReader`:

```

import java.io.IOException;
import java.util.jar.Attributes;
import java.util.jar.JarFile;

public class JarAttributeReader {
    private static final String OUTPUT_TEMPLATE = "%s=%s";

    // Метод считывает все атрибуты из jar-файла filename
    Attributes getAllAttributes(String filename) throws ␣
        IOException {
        JarFile jarFile;
        Attributes attributes;

        jarFile = new JarFile(filename);
        attributes = jarFile.getManifest().␣
            getMainAttributes();
        return attributes;
    }

    // Метод печатает в консоли атрибуты со значениями
    void printAllAttributesWithValues(String filename) {
        try {
            Attributes attributes = ␣
                getAllAttributes(filename);
            for (Object o : attributes.keySet()) {
                System.out.println ␣
                    (String.format ␣
                        (OUTPUT_TEMPLATE, ␣
                            o, attributes.getValue ␣
                                (o.toString())));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

В результате в консоли появится запись следующего вида:

```

Manifest-Version=1.0
Created-By=JarAttributeWriter
Attribute_1=Value_1
Attribute_2=Value_2

```

## Заключение

Следует отметить, что JAR API предоставляют довольно мощные средства для работы с JAR вообще и MANIFEST.MF, в частности, которые отнюдь не ограничиваются приведенными выше примерами. Ситуаций, когда правильное использования манифеста может существенно облегчить процесс отладки и/или распространения приложений, довольно много. Например, добавление версии приложения в качестве атрибута MANIFEST.MF позволит точно определить номер сборки приложения. Иными словами, в общем случае, в манифест можно записать различные параметры для последующего их использования при запуске приложения. 