GLOBAL 2007

Обучение Java

Тема **Безопасность в JAVA**

Часть **Загрузчики классов**

Автор **ASKIL (omendba@gmail.com)**

5.03.2007

С появлением технологии Java специалисты оценили по достоинству не только хорошо продуманные выразительные средства языка, но и механизмы обеспечения безопасности при выполнении апплетов, доставленных через Internet. Понятно, что доставка выполняемых апплетов имеет смысл только тогда, когда получатели уверены, что код не может нанести ущерб работе их компьютеров. Поэтому вопросы безопасности были и остаются основной заботой как разработчиков, так и пользователей технологии Java. Это означает, что, в отличие от других языков и систем, где безопасность обеспечивалась в последнюю очередь, механизмы защиты изначально являются неотъемлемой частью технологии Java. В технологии Java безопасность обеспечивают три механизма.

• Реализация языка (проверка границ массивов, выполнение только допустимых преобразований типов, отсутствие указателей и т.д.).

• Средства контроля доступа, которые проверяют выполняемые кодом действия (доступ к файлам, передача данных по сети и т.д.).

• Цифровая подпись, благодаря которой авторы кодов могут использовать стандартные алгоритмы аутентификации своих продуктов. Таким образом, пользователи могут точно определить, кто создал код и был ли код изменен после его подписания.

Виртуальная машина Java проверяет наличие неверных ссылок, выхода за границы массивов и т.п. Кроме того, контролируются объекты, отправляемые виртуальной машине Java.

Во время загрузки файлов классов в виртуальную машину проверяется их целостность.

Для максимального обеспечения безопасности используемый по умолчанию загрузчик классов и пользовательский загрузчик должны взаимодействовать с диспетчером, защиты (security manager), который контролирует выполняемые операции.

Загрузчики классов

Компилятор Java преобразует исходный код в код на внутреннем языке гипотетической виртуальной машины. Код, предназначенный для выполнения под управлением виртуальной машины, хранится в файлах классов с расширением .class. Каждый файл класса содержит определение и реализацию одного класса или интерфейса. Эти файлы интерпретируются программой, которая способна преобразовать набор инструкций виртуальной машины в машинные команды используемого компьютера.

Учтите, что интерпретатор виртуальной машины загружает только те файлы классов, которые необходимы для выполнения программы в данный момент. Допустим, что выполнение программы начинается с файла MyProgram.class. Ниже описаны действия, которые выполняет виртуальная машина Java.

1. В виртуальной машине имеется механизм загрузки файлов классов, например путем считывания их с диска или копирования по сети. С помощью этого механизма загружается содержимое файла класса MyProgram.

2. Если в классе MyProgram имеются переменные экземпляра, которые ссылаются на объекты других типов, то соответствующие файлы классов также загружаются. (Процесс загрузки всех классов, от которых зависит данный класс, называется разрешением класса.)

3. Затем виртуальная машина выполняет метод main() класса MyProgram. (Этот метод является статическим, поэтому никаких экземпляров класса MyProgram создавать не требуется.)

4. Если для работы метода main() или метода, вызываемого из main(), требуются дополнительные классы, то загружаются соответствующие им файлы.

Следует заметить, что механизм загрузки классов использует не один, а несколько загрузчиков. С каждой программой Java используется по крайней мере три загрузчика классов:

• первичный загрузчик классов (bootstrap class loader);

• расширенный загрузчик классов (extension class loader);

• системный загрузчик классов (system class loader), иногда называемый прикладным загрузчиком классов (application class loader).

Первичный загрузчик классов загружает системные классы, как правило, из файла rt.jar. Он является неотъемлемой частью виртуальной машины и обычно реализуется на языке C. Первичному загрузчику классов не соответствует ни один объект ClassLoader. Например, показанный ниже вызов возвращает значение null.

```
String.class.getClassLoader()
```

Расширенный загрузчик классов загружает стандартное расширение из каталога jre/lib/ext. Если переместить файлы .jar в этот каталог, то расширенный загрузчик классов найдет в нем нужные классы расширения даже без указания пути к классам. (Некоторые специалисты рекомендуют использовать этот механизм, чтобы избежать путаницы с указанием пути к классам, но такой способ связан с определенными сложностями, которые рассматриваются далее в разделе.)

Системный загрузчик классов загружает прикладные классы, которые размещаются в каталогах и файлах формата JAR/ZIP. Путь к классам должен быть указан с помощью переменной окружения CLASSPATH или параметра командной строки – classpath.

В системе, реализованной компанией Sun, расширенный и системный загрузчики реализуются на языке Java. Оба они представляют собой экземпляры класса URLClassLoader.

Помещать библиотеки в каталог jre/lib/ext нежелательно по следующей причине. Предположим, что вам понадобился один из классов этой библиотеки, который, в свою очередь, должен загружать класс, не являющийся сис-

темным или расширенным. Расширенный загрузчик классов не использует информацию о пути к классам, устанавливаемую в переменной окружения CLASSPATH. Этот факт необходимо учитывать, принимая решения об использовании каталога jre/lib/ext в качестве хранилища для своих классов.

Существует еще один аргумент против расположения JAR-файлов в каталоге jre/lib/ext. Нередко программисты забывают о файлах, с которыми они имели дело несколько месяцев назад. И не исключено, что им придется долго ломать голову, выясняя, почему же загрузчик классов игнорирует содержимое переменной окружения CLASSPATH. Причина же в том, что он загружает давно забытый класс из jre/lib/ext.

Загрузчики классов связаны отношениями родитель/потомок. У каждого загрузчика классов, за исключением первичного, имеется родительский загрузчик классов. Предполагается, что загрузчик классов дает шанс своему родителю загрузить любой нужный класс и загружает его сам только в том случае, если этого не может сделать родитель. Например, если системный загрузчик классов запрашивает загрузку системного класса (например, java.util.arrayList), то прежде он предлагает загрузить его расширенному загрузчику классов. Расширенный загрузчик классов, в свою очередь, предлагает сделать это первичному загрузчику классов. Наконец, первичный загрузчик классов находит и загружает класс из файла rt.jar; при этом все другие загрузчики классов прекращают дальнейший поиск.

Создавая загрузчик классов, необходимо предусмотреть делегирование загрузки классов родительскому загрузчику. В противном случае возникает потенциальная угроза безопасности системы. Написанный вами пользовательский загрузчик может загрузить системный класс, обойдя защиту.

Апплеты, сервлеты и RMI-заглушки загружаются пользовательскими загрузчиками классов. В особых случаях можно создать собственный загрузчик, который предоставит возможность проводить специализированную проверку байтового кода перед его передачей виртуальной машине. Например, можно создать собственный загрузчик, который сможет отказывать в загрузке класса, не помеченного знаком "оплачено".

Как правило, разработчику приложений не приходится беспокоиться о загрузчиках классов. Большинство классов загружаются потому, что этого требуют другие классы. Этот процесс остается прозрачным для программиста.

При использовании метода Class.forName() новый класс загружается посредством того же загрузчика, что и код, из которого был вызван этот метод. В большинстве случаев такой подход полностью устраивает программиста, однако он становится непригодным в следующих ситуациях.

• При реализации библиотечного класса, содержащего метод, который вызывает Class.forName().

• Если метод вызывается из класса приложения, причем этот класс и библиотечный класс были загружены разными загрузчиками.

• Если загруженный класс не виден для загрузчика, который загрузил библиотечный класс.

В этом случае в библиотечном классе необходимо выполнить дополнительную работу и извлечь прикладной загрузчик классов.

```
Thread t = Thread.currentThread();
ClassLoader loader = t.getContextClassLoader();
Class cl  = loader.loadClass(className);
```

Использование загрузчиков классов в качестве пространств имен

Каждый программист знает, что для исключения конфликтов, возникающих при использовании одинаковых имен для разных классов, применяются пакеты. Стандартная библиотека Java включает в себя два класса с одинаковым коротким именем Date и разными полными именами - java.util.Date и java.sql.Date. Короткие имена предназначены только для создания кода, более удобного для восприятия, и требуют добавления соответствующих операторов import. В выполняющейся программе все имена классов содержат имена своих пакетов.

Как ни странно, но в рамках одной и той же виртуальной машины могут существовать два разных класса, которые имеют одно и то же имя класса и пакета. Дело в том, что класс определяется по его полному имени и загрузчику класса. Такая технология весьма практична при загрузке кода из нескольких источников. Например, Web-браузер использует отдельные экземпляры загрузчика классов апплетов для каждой Web-страницы. Это позволяет виртуальной машине различать классы из разных документов независимо от их имен.

Существуют и другие способы использования данной технологии, например "горячая доставка" сервлетов и компонентов Enterprise JavaBeans.

Создание собственного загрузчика классов

Чтобы создать собственный загрузчик классов, следует просто расширить класс ClassLoader и переопределить метод findClass (String className).

Метод loadClass() суперкласса ClassLoader отвечает за делегирование родителю функций загрузки классов и вызывает метод findClass() в том случае, если класс еще не загружен или если родительский загрузчик классов не смог его загрузить.

В ранних версиях JDK программистам приходилось переопределять метод loadClass(), но теперь этого делать не рекомендуется.

Для реализации метода необходимо выполнить следующие действия.

1. Загрузить байтовый код класса из локальной файловой системы или из какого-то иного источника.

2. Вызвать метод defineClass() суперкласса ClassLoader для представления байтового кода виртуальной машине.

Рассмотрим пример реализации задачи. У нас имеется четыре класса. Класс NewClass2 выводит список файлов согласно выбранной папке. Его код изображен ниже:

```
import java.io.*;
public class NewClass2
{
    public NewClass2(String disk)
    {
      disk+="/";
      File f = new File(disk);
      File [] dir = f.listFiles();
      for (int i=0; i<dir.length; i++)
          System.out.println(dir[i]);
    }
}
```

Данный класс не имеет точек входа.

Класс NewClass имеет два главных метода, позволяющих ему функционировать:

```
public class NewClass
{
    public static void Noi(String s)
    {
      NewClass2 nc2 = new  NewClass2(s);
    }
    public static void main(String[] args)
    {
      System.out.println("Call main from NewClass");
    }
}
```

Данный класс может работать в автономном режиме, так как имеет точку входа.

Класс NewLoaderJavaClass переопределен в нашей программе и выполняет загрузку только класса NewClass. То есть другие классы он не может физически загрузить:

```
import java.io.*;
class NewLoaderJavaClass extends ClassLoader
{
    ////////////////////////Байты класса//////////////////////////////////////
    private byte [] LoadClassBytes(String path) throws Exception
    {
      FileInputStream fin = new FileInputStream(path+"/NewClass.class");
      byte [] b = new byte[fin.available()];
      fin.read(b);
      fin.close();
      return b;
    }
    ////////////////////////Поиск класса//////////////////////////////////////
    protected Class findClass(String name) throws ClassNotFoundException
     {
      try
      {
       byte [] classBytes = LoadClassBytes(name);
       Class cl = defineClass
              ("NewClass",classBytes,0,classBytes.length);
       return cl;
      }
      catch(Exception e){return null;}
     }
}
```

Реализация класса Main:

```
import java.lang.*;
import java.lang.reflect.*;
public class Main
{
    public static void main(String[] args)  throws Exception
    {
     if(args[0].equals("-part1"))
     {
       ClassLoader cl = new NewLoaderJavaClass();
       Class c = cl.loadClass(args[1]);
       Method [] m = c.getMethods();
       for (int i=0; i<m.length; i++)
       {
         if (m[i].getName().equals("Noi"))
         {
              String s=args[2]; //Содержимое папки
              m[i].invoke(null, (Object) s);
         }
       }
     }
     if(args[0].equals("-part2"))
     {
       ClassLoader cl = new NewLoaderJavaClass();
       Class c = cl.loadClass(args[1]);
       String s = args[2];
       Method m = c.getMethod("Noi", new Class[] { s.getClass() });
       m.invoke(null, (Object) s);
     }
    }
}
```

Пусть класс Main находится в папке “c:\”. А остальные классы в папке “c:\test”. Тогда перед вызовом главного класса необходимо установить переменные окружения:

```
set CLASSPATH=c:\test;c:\
```

Результат работы программы:

```
C:\>java Main -part1 c:/test c:
c:\AUTOEXEC.BAT
c:\boot.ini
c:\Bootfont.bin
c:\CONFIG.SYS
c:\Documents and Settings
c:\hiberfil.sys
c:\IO.SYS
c:\Main.class
c:\MSDOS.SYS
c:\NTDETECT.COM
c:\ntldr
c:\pagefile.sys
c:\Program Files
c:\RECYCLER
c:\System Volume Information
c:\test
c:\WINDOWS
```

```
C:\>java Main -part1 c:/test c:/test
c:\test\NewClass.class
c:\test\NewClass2.class
c:\test\NewLoaderJavaClass.class
```

```
C:\>java Main -part2 c:/test c:/java
c:\java\hello.txt
```