2008 г.

[Пишем свой загручик java-классов](http://samolisov.blogspot.com/2008/01/java.html)

Продолжаю изучать как устроена и работает Java-машина. Тем более что при приеме на работу очень любят спрашивать различные [тонкости и вкустности](http://samolisov.blogspot.com/2007/11/blog-post_18.html). В одной из предыдущих заметок я уже писал про то, как [создаются объекты в Java](http://samolisov.blogspot.com/2007/11/java.html). Пришло время разобраться из чего они создаются.  
  
  
**Введение**  
Как известно, Java как платформа состоит из двух частей: компилятора исходного кода в байт-код и интерпретатора байткода (я знаю что в некоторых реализациях - компилятора, но мы рассмотрим общий случай). Понятно, что прежде чем некий код интерпретировать его необходимо загрузить в ОЗУ компьютера. В Java реализована модель так называемой **динамической загрузки классов** (она же - **модель позднего связывания**). Динамическая загрузка классов в Java обладает следующими особенностями:  
  
1. ***Отложенная (lazy) загрузка.*** Каждый класс загружается в память только при необходимости.  
  
2. ***Проверка корректности загружаемого кода*** (type safeness). Все действия, связанные с проверкой валидности кода выполняются на этапе загрузки.  
  
3. ***Программируемая загрузка***. Программист может написать свои загрузчики классов, которые подгружают классы например из БД, архива, по сети или еще каким-либо образом.  
  
4. ***Множественные пространства имен***. Каждый загрузчик имеет свое пространство имен для создаваемых классов. Т.е. если классы одинаковы и находятся в одном пакете, но загружаются разными загрузчиками - они считаются разными.  
  
Очень подробно модель динамической загрузки и модель делегирования загрузки описаны в статье на [сайте питерских тестировщиков JVM](http://blogs.sun.com/vmrobot/entry/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B_%D0%B4%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9_%D0%B7%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_%D0%B2). Я считаю нецелесообразным перепечатывать эту статью, но настоятельно рекомендую прочесть ее, чтобы понимать дальнейшее.  
  
**Пишем свой загрузчик**  
  
Собственно лучше один раз попробовать, чем десять - прочитать. Поэтому попробуем написать загрузчик классов из Jar-файла. Я знаю что системный загрузчик умеет загружать классы из jar-файлов, но только тех, которые указаны в *CLASSPATH*. Что же делать, если jar-файл не указан в *CLASSPATH*, например это - плагин? Вот загрузчик, решающий эту проблему, мы и напишем.  
  
Но прежде чем писать, необходимо продумать алгоритм загрузчика и требования к нему. Поэтому разберемся как работает системный загрузчик.  
  
Вообще в начале работы приложения создаются три загрузчика:

* базовый загрузчик (*bootstrap/primodial class loader)*
* загрузчик расширений (*extention class loader*)
* системный загрузчик (*system/application class loader*)

Собственно наиболее важен для нас именно системный загрузчик, именно он загружает классы из *CLASSPATH*. Загрузка классов происходит по следующему алгоритму:  
  
1.  Поиск в списке ранее загруженных классов. Проверяется, запрашивался ли данный класс ранее. В случае, если запрашивался загружается из кэша.  
  
2. Если класс ранее не запрашивался, то производится попытка делегирования родительскому загрузчику. Это позволяет загружать классы тем загрузчиком, который находится ближе всего к базовому в иерархии делегирования.  
  
3. Загружаем класс сами. Если родительский загрузчик не смог загрузить запрошенный класс, текущий загрузчик сам производит процесс загрузки требуемого класса - находит байт-код и на его основе создает класс.  
  
Вообще пользовательский загрузчик не обязан работать по данному алгоритму, но в таком случае возможны проблемы, связанные с ограничением области видимости загружаемых классов.  
  
В этом примере загружаются ВСЕ файлы из jar-архива, даже если они не вызываются:

**package** org.beq.classloader;  
  
**import** java.io.[**IOException**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/IOException.html);  
**import** java.io.[**InputStream**](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+InputStream);  
**import** java.util.[**Enumeration**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Enumeration.html);  
**import** java.util.[**HashMap**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/HashMap.html);  
**import** java.util.jar.[**JarEntry**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarEntry.html);  
**import** java.util.jar.[**JarFile**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarFile.html);  
  
/\*\*  
 \* Загружаем файлы из заданного jar-архива  
 \* Классы должны относится к заданному пакету - пример валидации при загрузке  
 \*  
 \* @author Pavel  
 \*  
 \*/  
  
**public** **class** JarClassLoader **extends** [**ClassLoader**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ClassLoader.html) {  
  
    **private** HashMap<String, Class<?>> cache = **new** HashMap<String, Class<?>>();  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) jarFileName;  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) packageName;  
    **private** **static** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) WARNING = "Warning : No jar file found. Packet unmarshalling won't be possible. Please verify your classpath";  
  
    **public** JarClassLoader([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) jarFileName, [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) packageName) {      
  
        **this**.jarFileName = jarFileName;          
        **this**.packageName = packageName;  
        cacheClasses();  
    }  
  
    /\*\*  
     \* При создании загрузчика извлекаем все классы из jar и кэшируем в памяти  
     \*  
     \*/  
  
    **private** void cacheClasses() {  
  
        **try** {             
            [**JarFile**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarFile.html) jarFile = **new** [**JarFile**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarFile.html)(jarFileName);  
            Enumeration entries = jarFile.entries();  
  
            while (entries.hasMoreElements()) {  
                **[JarEntry](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarEntry.html)** jarEntry = (**[JarEntry](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarEntry.html)**) entries.nextElement();  
                // Одно из назначений хорошего загрузчика - валидация классов на этапе загрузки  
                if (match(normalize(jarEntry.getName()),packageName)) {                     
                  byte[] classData = loadClassData(jarFile,jarEntry);   
                    if (classData != null) {  
                        Class<?> clazz = defineClass(stripClassName(normalize(jarEntry.getName())),classData,0,classData.length);  
                        cache.put(clazz.getName(), clazz);  
                        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("== class "+clazz.getName()+" loaded in cache");  
                    }  
                }  
  
            }  
        }

**catch** (**[IOException](http://java.sun.com/j2se/1.5.0/docs/api/java/io/IOException.html)** IOE) {  
            // Просто выведем сообщение об ошибке  
            [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println(WARNING);  
        }  
    }  
        
    /\*\*  
     \* Собственно метод, который и реализует загрузку класса  
     \*  
     \*/  
  
    **public** **synchronized** Class<?> loadClass([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) name) **throws** **[ClassNotFoundException](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ClassNotFoundException.html)** {  
  
        Class<?> result = cache.get(name);  
       
        // Возможно класс вызывается не по полному имени - добавим имя пакета  
  
        if (result == null)  
            result = cache.get(packageName + "." + name);             
            
        // Если класса нет в кэше то возможно он системный  
  
        if (result == null)  
            result = **super**.findSystemClass(name);         
        
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("== loadClass(" + name + ")");         
         
        **return** result;  
    }  
  
    /\*\*  
     \* Получаем каноническое имя класса  
     \* @param className  
     \* @return  
     \*/  
  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) stripClassName([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) className) {  
  
        **return** className.substring(0, className.length() - 6);  
    }  
  
    /\*\*  
     \* Преобразуем имя в файловой системе в имя класса  
     \* (заменяем слэши на точки)  
     \*  
     \* @param className  
     \* @return  
     \*/  
  
    **private** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) normalize([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) className) {  
  
        **return** className.replace('/', '.');  
    }  
  
    /\*\*  
     \* Валидация класса - проверят принадлежит ли класс заданному пакету и имеет ли  
     \* он расширение .class  
     \*    
     \* @param className  
     \* @param packageName  
     \* @return  
     \*/  
  
    **private** boolean match([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) className, [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) packageName) {

        **return** className.startsWith(packageName) && className.endsWith(".class");  
    }     
     
    /\*\*  
     \* Извлекаем файл из заданного JarEntry  
     \*  
     \* @param jarFile - файл jar-архива из которого извлекаем нужный файл  
     \* @param jarEntry - jar-сущность которую извлекаем  
     \* @return null если невозможно прочесть файл  
     \*/  
  
    **private** byte[] loadClassData([**JarFile**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarFile.html) jarFile, [**JarEntry**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/jar/JarEntry.html) jarEntry) **throws** [**IOException**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/IOException.html) {  
       long size = jarEntry.getSize();     

      if (size == -1 || size == 0)  
            **return** null;  
        
        byte[] data = **new** byte[(int)size];  
        **[InputStream](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+InputStream)** in = jarFile.getInputStream(jarEntry);  
        in.read(data);  
         
        **return** data;  
    }  
}

Алгоритм работы данного загрузчика следующий: при создании загрузчика через конструктор вызывается метод *private void cacheClasses()*, который читает переданный jar-архив и загружает классы из него в кэш. Кэш представлен в виде *HashMap* где названию соответствует класс. Здесь же и происходит валидация - загружаются только те файлы, которые соответсвуют заданному пакету и имеют расширение *.class*  
  
**upd:**  
Метод *defineClass* служит для создания класса из загруженного байт-кода. Им осуществляется валидация загруженного байт-кода и анализ зависимостей.  
  
Через переопределенный метод *loadClass(String name)* происходит использование загрузчика. В данном случае метод довольно прост - если не можем загрузить класс из кэша - значит он системный и делегируем его загрузку базовому загрузчику.  
  
Вообще стоит заметить, что приведенный выше код неоптимален - в нем загружаются все классы из Jar, даже если они не вызываются. Но он очень хорошо демонстрирует создание загрузчика.

**Использование созданного загрузчика**  
  
Одним из наиболее сложных моментов является использование созданного загрузчика. Вообще загрузчик можно использовать тремя способами: указава его как базовый для всего приложения через параметр командной строки ***-Djava.system.class.loader***. Тем самым ваш загрузчик станет ***загрузчиком по-умолчанию***.  
  
Более подробно хочется рассмотреть способ программного указания загрузчика. Пусть наш загрузчик не является загрузчиком по-умолчанию. Тогда создать объект с его помощью можно следующим образом:

// Создаем загрузчик  
  
JarClassLoader jarClassLoader = **new** JarClassLoader("beq.jar", "org.beq.classloader.classes.impl");  
  
// Загружаем класс  
Class<?> clazz = loadClass("JarSample"); //наверное при таком методе загрузки класса(loadClass()) это метод автоматически подтявается из загрузчика объект которого мы только, что создали. Т.к. если бы это был метод forName(), то загрузчик пришлось бы явно указать загрузчик.  
  
// Создаем экземпляр класса  
IJarSample sample = (IJarSample) clazz.newInstance();  
  
sample.demo("Test");

Класс JarSample и интерфейс IJarSample имеют следующий код:

**package** org.beq.classloader.classes.impl;  
  
**import** org.beq.classloader.classes.IJarSample;  
  
/\*\*  
 \* Демонстрационный класс, будет загружаться из Jar  
 \* @author Pavel  
 \*  
 \*/  
  
**public** **class** JarSample **implements** IJarSample {  
  
    **public** JarSample() {  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("JarSample::JarSample()");  
    }  
     
    **public** void demo([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) str) {  
  
        JarClass2Samle s = **new** JarClass2Samle();  
        s.demo();  
         
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("JarSample::demo(String str)");       
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println(str);  
    }  
}

Написано через интерфейс и наследующий его класс, чтобы к определенным в классе методам интерфейса можно было обращаться явно, а не через invoke().

**package** org.beq.classloader.classes;

/\*\*  
 \* Интерфейс к загружаемому классу  
 \*  
 \* @author Pavel  
 \*  
 \*/  
  
**public** **interface** IJarSample {  
    **public** void demo([**String**;](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) str);  
}

Наиболее важно в данном коде то, что **интерфейс обязательно должен загружаться системный загрузчиком**. Как я уже говорил, классы загруженые разными загрузчиками для системы разные. Если интерфейс *IJarSample* будет загружаться нами написанным загрузчиком мы не сможем его использовать в классах, загружаемых системным загрузчиком - будем получать *ClassCastException*.  
  
И, напоследок, хотелось бы отметить еще одну особенность: если класс загружается сторонним загрузчиком, то и ***все классы на которые он ссылается*** (например *JarClass2Samle*) ***будут загружаться этим же загрузчиком***. Собственно данное правило вытекает из правила различных областей видимости - если бы класс загружался другим загрузчиком (например системным) он был бы просто недоступным.  
  
Изложенная выше информация целиком и полностью основана на личных экспериментах и наблюдениях и не претендует на истину в последней инстанции. Я буду очень рад всем Вашим коментариям, особенно замечаниями и исправлениям допущенных неточностей.  
  
  
**upd:**  
  
**Еще один пример загрузчика и его правильного использования.**  
**Где наш Загрузчик любой загружаемый им класс будет проверять в списке – если нашел загрузит собой, нет загрузит системным загрузчиком.**

Интерфейс:

**package** sexypackage;  
  
**public** **interface** ISexyInterface {  
    **public** void makeBar ();  
}

Демо-класс, который будет грузится загрузчиком

**package** sexypackage;  
  
**public** **class** SexyClassForLoader **implements** ISexyInterface {  
  
    **public** **static** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) stat\_foo = "hello stat\_foo";  
  
    **static** {  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("SexyClassForLoader$$static");  
    }  
  
    **public** SexyClassForLoader {  
         [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("SexyClassForLoader$$init");  
    }  
  
    **public** **static** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) getStatFoo() {  
         **return** stat\_foo;  
    }  
  
    **public** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) getSimpleFoo() {  
        **return** simple\_foo;  
    }  
  
    **public** [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) simple\_foo = "hello simple\_foo";  
  
    **public** void makeBar() {  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println ("make bar");  
    }  
}

Как видем класс и интерфейс лежат в одном пакете, но мы сделаем так, чтобы они грузились разными загрузчиками.  
  
Собственно загрузчик:

**package** jmxtest.test;  
  
**import** java.io.[**File**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/File.html);  
**import** java.io.[**FileInputStream**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileInputStream.html);  
**import** java.io.[**IOException**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/IOException.html);  
**import** java.util.[**HashMap**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/HashMap.html);  
  
/\*\*  
\* Класс-загрузчик  
\*/  
  
**class** XLoader **extends** [**ClassLoader**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ClassLoader.html)  
{  
    // карта отображения имен классов на файлы .class, где хранятся их определения  
  
    HashMap mappings;  
  
    XLoader(HashMap mappings)  
    {  
        **this**.mappings = mappings;  
    }  
  
    **public** **synchronized** Class loadClass([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) name) **throws** [**ClassNotFoundException**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ClassNotFoundException.html)  
    {  
        **try**  
        {  
            [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("loadClass (" + name + ")");  
             
            // важно!  
            // приоритет отдан именно загрузке с помощью встроенного загрузчика  
            if (!mappings.containsKey(name))  
            {  
                **return** **super**.findSystemClass(name);  
            }  
             
            [**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) fileName = mappings.get(name);  
            **[FileInputStream](http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileInputStream.html)** fin = **new** **[FileInputStream](http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileInputStream.html)**(fileName);              
            byte[] bbuf = **new** byte[(int)(**new** [**File**](http://java.sun.com/j2se/1.5.0/docs/api/java/io/File.html)(fileName).length())];  
            fin.read(bbuf);  
            fin.close();  
             
            **return** defineClass(name, bbuf, 0, bbuf.length);  
        }  
        **catch** (**[IOException](http://java.sun.com/j2se/1.5.0/docs/api/java/io/IOException.html)** e)  
        {  
            e.printStackTrace();  
            **throw** **new** **[ClassNotFoundException](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ClassNotFoundException.html)**(e.getMessage(), e);  
        }  
    }  
}

Как видим приоритет в данном загрузчике отдан именно загрузке с помощью системного загрузчика. Если класса нет в списке загрузки, то он будет загружен с помощью системного загрузчика. Формирование списка загрузки производится в вызывающей программе.  
  
Ну и собственно пример вызывающей программы:

**package** jmxtest.test;  
  
**import** java.util.[**HashMap**](http://java.sun.com/j2se/1.5.0/docs/api/java/util/HashMap.html);  
**import** sexypackage.ISexyInterface;  
  
**public** **class** Loader  
{  
    **public** **static** void main([**String**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html)[] args) **throws** [**Exception**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Exception.html)  
    {  
        HashMap mappings = **new** HashMap(); //создаем коллекцию  
        mappings.put("sexypackage.SexyClassForLoader", "E:\\@Pavel\\MyTeach\\ClassLoader\\bin\\sexypackage\\SexyClassForLoader.class"); //заполняем нашу коллекцию данными, среди которых будем проверять на наличие   
  
        // Если убрать комментарий - будет больно  
        /\*  
            mappings.put("sexypackage.ISexyInterface",  
            "путь\\classes\\sexypackage\\ISexyInterface.class");  
        \*/  
  
        XLoader xloa = **new** XLoader(mappings); //создаем новый объект нашего загрузчика  
        Class sexy\_cla = xloa.loadClass("sexypackage.SexyClassForLoader"); //загрузит класс либо нашим загрузчиком либо системным, в зависимости от наличия в списке  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("class was loaded");  
        **[System](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html)**.out.println("begin object creation");  
         
        [**Object**](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+Object) sexy\_ob = sexy\_cla.newInstance(); //создаем объект объект загруженного класса  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("object was created");  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("invoke: getFoo" + sexy\_cla.getMethod("getSimpleFoo").invoke(sexy\_ob));  
        [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("get: stat\_foo" +

sexy\_cla.getField("stat\_foo").get(sexy\_ob));  
//это были вызовы метода/поля, которых нет в интерфейсе (они были добавлены в классе) через Reflection (с использованием метода invoke())

        ISexyInterface local\_sexy = (ISexyInterface) sexy\_ob; //это «финт ушами», связывающий интерфейс ISexyInterface, запущенный системным загрузчиком (т.к. проверку на наличие в коллекции “mappings” он не прошел) и нашего класса для того, чтобы в следующей строке кода мы могли вызвать нужный нам метод “makeBar()” напрямую, а не через “invoke()”.  
        local\_sexy.makeBar(); //вызов метода, который есть в интерфейсе  
    }  
}

Данный пример демонстрирует загрузку класса с помощью нашего класс-лоадера, порядок вызова блока статической и обычной инициализации, а также вызов методов класса через reflection и через интерфейс.  
  
Результат работы программы:  
***loadClass (sexypackage.SexyClassForLoader)  
loadClass (sexypackage.ISexyInterface)  
loadClass (java.lang.Object)  
class was loaded  
begin object creation  
loadClass (java.lang.System)  
loadClass (java.io.PrintStream)  
SexyClassForLoader$$static  
SexyClassForLoader$$init  
object was created  
loadClass (java.lang.String)  
invoke: getFoohello simple\_foo  
get: stat\_foohello stat\_foo  
make bar***  
  
Если раскомментировать строки добавляющие интерфейс в карту вызова, то получим ошибку. Получится, что интерфейс будет грузится не системным, а нашим загрузчиком и не будет доступен в классе *Loader*.