

Оптимизация загрузки классов

Максим Парыгин

В этой статье вы познакомитесь с оптимизацией загрузки классов с помощью базы данных.

По умолчанию JRE загружает классы посредством специальных классов — загрузчиков (classloaders). Происходит это следующим образом. У загрузчика класса запрашивается (например, посредством метода **loadClass**) экземпляр класса Class для необходимого класса. Загрузчик ищет класс в jar файлах, указанных в командной строке, и в файловой системе. Если необходимый файл с расширением .class будет найден — загрузчик вернет созданный по файлу экземпляр оболочки класса (Экземпляр Class), если нет — выбросит исключение.

Теперь займемся подсчетами. Каждый jar файл требует распаковки (если сжат). Время уходит на поиск файла, на его извлечение. Тут же выявляется еще один эффект. Поиск происходит последовательно в jar, в том порядке который был задан с командной строки. В командной строке их может быть порядка одного — двух десятков. Еще более тяжелая ситуация возникает когда приходится производить поиск в файловой системе — тут поиск может затянуться на десятки доли секунды.

Простейшим выходом может оказаться использование базы данных — такой же подход используется Oracle для загрузки классов. В простейшем случае необходимо создать таблицу в базе данных. Таблица должна иметь поле наименования (полного имени класса включая пакеты) — индексированного и уникального поля, и поле blob для хранения непосредственно байт-кода. Нам еще понадобится новый **classloader** умеющий работать с нашей базой. В случае использования базы как носителя классов загрузка происходит следующим образом. При запросе класса происходит обращение к базе данных. База данных производит поиск в индексе необходимого имени. Выбирается байтовый массив, из которого и формируется обложка класса. В большинстве случаев поиск в одной таблице будет произведен гораздо быстрее

множества поисков в файлах и файловой системе. В доверок ко всему база байт-кода может использоваться несколькими клиентами — что может, например, применяться для централизованного управления версиями приложения.

1. Теперь практика

- А) База данных. Простейшим, но не самым худшим, будет использование базы данных MySQL. Простые запросы будут обрабатываться достаточно быстро. Предположим что мы используем MySQL (для других баз данных изменений практически не будет). Создаем базу данных, например, class.

```
create database class; use class;
```

Создаем таблицу с именем classes.

```
create table classes (name char(255) not null,
    value blob not null, primary key(name));
```

Возможно также использовать более сложный вариант — создание 2 таблиц — таблицы байт кода, и таблицы с именами jar файлов (в таблице байт-кода необходимо будет сделать ссылку на таблицу jar файлов). Также не забудьте завести в базе пользователя и установить ему пароль.

- Б) Загрузчик классов (/data/jprojects/javable/src/sqlClassLoader.java).

```
import java.util.Hashtable;
import java.sql.*;

/**
 * Загрузчик классов из базы данных
 */

public class sqlClassLoader extends ClassLoader {

    Hashtable cache = new Hashtable();

    /**
     * Обращение к базе данных
     * @param name
     * @return */
```

Оптимизация загрузки классов

```
private byte[] loadClassData(String name) {
    try {
        byte[] result = null;
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://java.kkb.kz/class",
            "server", "52fgab");
        PreparedStatement stmt =
            conn.prepareStatement("SELECT value FROM classes
                WHERE name = ?");

        stmt.setString(1, name);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            result = rs.getBytes(1);
        }
        rs.close();
        stmt.close();
        conn.close();
        return result;
    } catch (SQLException e) {
        //To change body of catch statement use
        //Options | File Templates.
        e.printStackTrace();
    }
}
return null;
}

/**
 * Получить класс из базы данных
 * @param name
 * @return * @throws ClassNotFoundException
 */
public synchronized Class loadClass(String name) throws ClassNotFoundException {
    // Получить класс из кэша
    Class c = (Class) cache.get(name);
    if (c != null) return c;
    // В кэше класс не обнаружен -
    // ищем класс в базе данных
    byte data[] = loadClassData(name);
    if (data == null) {
```

```
// Класс в базе данных не обнаружен –
// выбрасываем исключение
    throw new ClassNotFoundException();
} else {
    // Класс обнаружен
    c = defineClass(data, 0, data.length);
    cache.put(name, c);
    return c;
}
}

/**
 * Статическая инициализация драйвера базы данных
 */
static {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
}
```

- **В) Тестовый пример** (/data/jprojects/ javable/src/sample.java).

```
/**
 * Загрузчик классов из базы данных
 */
public class sample {

    /**
     * Главный метод
     * @param args
     */
    public static void main(String[] args) throws Exception {

        // Создаем загрузчик
        sqlClassLoader sql = new sqlClassLoader();
        // Загружаем класс
```

Оптимизация загрузки классов

```
Class a = sql.loadClass("sample.class");  
// Создаем экземпляр класса  
Object instance = a.newInstance();  
  
// .....  
  
}  
}
```

2. Заключение

Наш загрузчик обладает несколькими очевидными недостатками. Самый существенный — для загрузки каждого класса создается отдельное соединение с базой. Можно использовать постоянное соединение или использовать пул соединений с базой данных (при использовании пула не забудьте сделать метод **loadClass** асинхронным — если Вы будете использовать многопоточную загрузку классов). Также можно инициализировать параметры соединения с базой в конструкторе загрузчика. Еще одним существенным недостатком является явная загрузка — вместо неявной через статические методы класса **Class**. Для этого придется переделать загрузчик, чтобы он в случае ненахождения класса пытался загрузить класс следующим загрузчиком. Также, в командной строке нужно будет указать опцию **-Djava.system.class.loader=sqlClassLoader** чтобы JVM использовала ваш загрузчик первым. Помимо этого придется справиться с ситуацией рекурсивного вызова драйвера самого себя, когда драйверу при инициализации понадобятся определенные классы. И последнее — наш загрузчик не загружает двоичные ресурсы — добавить эту возможность не составит никакого труда.

3. Ресурсы

- [MySQL](#)