允许你的包名以"java."开头

下面介绍如何突破 JDK 不允许自定义的包名以"java."开头这一限制。这一技巧对于基于已有的 JDK 向 java.*中添加新类还是有所帮助的。

无论是经验丰富的 Java 程序员，还是 Java 的初学者，总会有一些人或有意或无意地创建一个包名为"java"的类。但出于安全方面的考虑，JDK 不允许应用程序类的包名以"java"开头，即不允许 java，java.foo 这样的包名。但 javax，javaex 这样的包名是允许的。

1. 例子

比如，以 OpenJDK 8 为基础，臆造这样一个例子。笔者想向 OpenJDK 贡献一个同步的 HashMap，即类 SynchronizedHashMap，而该类的包名就为 java.util。SynchronizedHashMap 是 HashMap 的同步代理，由于这两个类是在同一包内，SynchronizedHashMap 不仅可以访问 HashMap 的 public 方法与变量，还可以访问 HashMap 的 protected 和 default 方法与变量。SynchronizedHashMap 看起来可能像下面这样：

```
package java.util;

public class SynchronizedHashMap<K, V> {

   private HashMap<K, V> hashMap = null;

   public SynchronizedHashMap(HashMap<K, V> hashMap) {
      this.hashMap = hashMap;
   }

   public SynchronizedHashMap() {
      this(new HashMap<>());
   }

   public synchronized V put(K key, V value) {
      return hashMap.put(key, value);
   }

   public synchronized V get(K key) {
      return hashMap.get(key);
   }

   public synchronized V remove(K key) {
      return hashMap.remove(key);
   }
```

```
    public synchronized int size() {
        return hashMap.size; // 直接调用 HashMap.size 变量，而非 HashMap.size()方法
    }
}
```

2. ClassLoader 的限制

使用 javac 去编译源文件 SynchronizedHashMap.java 并没有问题，但在使用编译后的 SynchronizedHashMap.class 时，JDK 的 ClassLoader 则会拒绝加载 java.util.SynchronizedHashMap。

设想有如下的应用程序：

```
import java.util.SynchronizedHashMap;

public class SyncMapTest {

    public static void main(String[] args) {
        SynchronizedHashMap<String, String> syncMap = new SynchronizedHashMap<>();
        syncMap.put("Key", "Value");
        System.out.println(syncMap.get("Key"));
    }
}
```

使用 java 命令去运行该应用时，会报如下错误：

```
Exception in thread "main" java.lang.SecurityException: Prohibited package name: java.util
    at java.lang.ClassLoader.preDefineClass(ClassLoader.java:659)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:758)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:467)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:368)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:362)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:361)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    at SyncMapTest.main(SyncMapTest.java:6)
```

方法 ClassLoader.preDefineClass()的源代码如下：

```
private ProtectionDomain preDefineClass(String name,
        ProtectionDomain pd)
{
```

```
    if (!checkName(name))
        throw new NoClassDefFoundError("IllegalName: " + name);

    if ((name != null) && name.startsWith("java.")) {
        throw new SecurityException
            ("Prohibited package name: " +
            name.substring(0, name.lastIndexOf('.')));
    }
    if (pd == null) {
        pd = defaultDomain;
        }

    if (name != null) checkCerts(name, pd.getCodeSource());

    return pd;
}
```

很清楚地，该方法会先检查待加载的类全名(即包名+类名)是否以"java."开头，如是，则抛出SecurityException。那么可以尝试修改该方法的源代码，以突破这一限制。

从 JDK 中的 src.zip 中拿出 java/lang/ClassLoader.java 文件，修改其中的 preDefineClass 方法以去除相关限制。重新编译 ClassLoader.java，将生成的 ClassLoader.class，ClassLoader$1.class，ClassLoader$2.class，ClassLoader$3.class，ClassLoader$NativeLibrary.class，ClassLoader$ParallelLoaders.class 和 SystemClassLoaderAction.class 去替换 JDK/jre/lib/rt.jar 中对应的类。

再次运行 SyncMapTest，却仍然会抛出相同的 SecurityException，如下所示：

```
Exception in thread "main" java.lang.SecurityException: Prohibited package nam
e: java.util
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:760)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:467)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:368)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:362)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:361)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    at SyncMapTest.main(SyncMapTest.java:6)
```

此时是由方法 ClassLoader.defineClass1()抛出的 SecurityException。但这是一个 native 方法，那么仅通过修改 Java 代码是无法解决这个问题的(JDK 真是层层设防啊)。原来在 Hotspot 的 C++源文件 hotspot/src/share/vm/classfile/systemDictionary.cpp 中有如下语句：

```
const char* pkg = "java/";
if (!HAS_PENDING_EXCEPTION &&
    !class_loader.is_null() &&
    parsed_name != NULL &&
    !strncmp((const char*)parsed_name->bytes(), pkg, strlen(pkg))) {
  // It is illegal to define classes in the "java." package from
  // JVM_DefineClass or jni_DefineClass unless you're the bootclassloader
  ResourceMark rm(THREAD);
  char* name = parsed_name->as_C_string();
  char* index = strrchr(name, '/');
  *index = '\0'; // chop to just the package name
  while ((index = strchr(name, '/')) != NULL) {
    *index = '.'; // replace '/' with '.' in package name
  }
  const char* fmt = "Prohibited package name: %s";
  size_t len = strlen(fmt) + strlen(name);
  char* message = NEW_RESOURCE_ARRAY(char, len);
  jio_snprintf(message, len, fmt, name);
  Exceptions::_throw_msg(THREAD_AND_LOCATION,
    vmSymbols::java_lang_SecurityException(), message);
}
```

修改该文件以去除掉相关限制，并按照本系列的第一篇文章中介绍的方法去重新构建一个 OpenJDK。那么，这个新的 JDK 将不会再对包名有任何限制了。

3. 覆盖 Java 核心 API？

开发者们在使用主流 IDE 时会发现，如果工程有多个 jar 文件或源文件目录中包含相同的类，这些 IDE 会根据用户指定的优先级顺序来加载这些类。比如，在 Eclipse 中，右键点击某个 Java 工程-->属性-->Java Build Path-->Order and Export，在这里调整各个类库或源文件目录的位置，即可指定加载类的优先级。

当开发者在使用某个开源类库(jar 文件)时，想对其中某个类进行修改，那么就可以将该类的源代码复制出来，并在 Java 工程中创建一个同名类，然后指定 Eclipse 优先加息自己创建的类。即，在编译时与运行时用自己创建的类去覆盖类库中的同名类。那么，是否可以如法炮制去覆盖 Java 核心 API 中的类呢？

考虑去覆盖类 java.util.HashMap，只是简单在它的 put()方法添加一条打印语。那么就需要将 src.zip 中的 java/util/HashMap.java 复制出来，并在当前 Java 工程中创建一个同名类 java.util.HashMap，并修改 put()方法，如下所示：

```
package java.util;

public class HashMap<K,V> extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable {

    ...
```

```
    public V put(K key, V value) {
        System.out.printf("put - key=%s, value=%s%n", key, value);
        return putVal(hash(key), key, value, false, true);
    }
    ...
}
```

此时，在 Eclipse 环境中，SynchronizedHashMap 使用的 java.util.HashMap 被认为是上述新创建的 HashMap 类。那么运行应用程序 SyncMapTest 后的期望输出应该如下所示：

```
put - key=Key, value=Value
Value
```

但运行 SyncMapTest 后的实际输出却为如下：

```
Value
```

看起来，新创建的 java.util.HashMap 并没有被使用上。这是为什么呢？能够"想像"到的原因还是类加载器。关于 Java 类加载器的讨论超出了本文的范围，而且关于该主题的文章已是汗牛充栋，但本文仍会简述其要点。

Java 类加载器由下至上分为三个层次：引导类加载器(Bootstrap Class Loader)，扩展类加载器(Extension Class Loader)和应用程序类加载器(Application Class Loader)。其中引导类加载器用于加载 rt.jar 这样的核心类库。并且引导类加载器为扩展类加载器的父加载器，而扩展类加载器又为应用程序类加载器的父加载器。同时 JVM 在加载类时实行委托模式。即，当前类加载器在加载类时，会首先委托自己的父加载器去进行加载。如果父加载器已经加载了某个类，那么子加载器将不会再次加载。

由上可知，当应用程序试图加载 java.util.Map 时，它会首先逐级向上委托父加载器去加载该类，直到引导类加载器加载到 rt.jar 中的 java.util.HashMap。由于该类已经被加载了，我们自己创建的 java.util.HashMap 就不会被重复加载。

使用 java 命令运行 SyncMapTest 程序时加上 VM 参数-verbose:class，会在窗口中打印出形式如下的语句：

```
[Opened /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
[Loaded java.lang.Object from /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
...
[Loaded java.util.HashMap from /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
[Loaded java.util.HashMap$Node from /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
...
[Loaded java.util.SynchronizedHashMap from file:/home/ubuntu/projects/test/classes/]
Value
[Loaded java.lang.Shutdown from /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
[Loaded java.lang.Shutdown$Lock from /home/ubuntu/jdk1.8.0_custom/jre/lib/rt.jar]
```

从中可以看出，类 java.util.HashMap 确实是从 rt.jar 中加载到的。但理论上，可以通过自定义类加载器去打破委托模式，然而这就是另一个话题了。