破坏双亲委派能改写String吗?

典型回答

java通过双亲委派模型保证了java核心包中的类不会被破坏，但破坏双亲委派能够脱离加载范围的限制，增强第三方组件的能力。

什么是双亲委派？如何破坏？

我们虽然可以通过破坏双亲委派屏蔽Bootstrap ClassLoader，但无法重写`java.`包下的类，如`java.lang.String`。

扩展知识

我们知道，要破坏双亲委派模型是需要`extends ClassLoader`并重写其中的`loadClass()`和`findClass()`方法。
之所以无法替换`java.`包的类，主要原因是即使我们破坏双亲委派模型，依然需要调用父类中（`java.lang.ClassLoader.java`）的`defineClass()`方法来把字节流转换为一个JVM识别的class。而`defineClass()`方法中通过`preDefineClass()`方法限制了类全限定名不能以`java.`开头。如下代码所示：

```
//将字节流转换成jvm可识别的java类
  protected final Class<?> defineClass(String name, byte[] b, int off, int
len,
                                       ProtectionDomain protectionDomain)
        throws ClassFormatError
    {
        protectionDomain = preDefineClass(name, protectionDomain);//检查类全限
定名是否有效
        String source = defineClassSourceLocation(protectionDomain);
        Class<?> c = defineClass1(name, b, off, len, protectionDomain,
source);//调用本地方法，执行字节流转JVM类的逻辑。
        postDefineClass(c, protectionDomain);
        return c;
    }

//检查类名的有效性
 private ProtectionDomain preDefineClass(String name,
                                         ProtectionDomain pd)
```

```
    {
        if (!checkName(name))
            throw new NoClassDefFoundError("IllegalName: " + name);
        if ((name != null) && name.startsWith("java.")) { //禁止替换以java.开头
的类文件
            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;
    }
```

注意，`defineClassX`三兄弟是三个本地方法，用于不同参数长度的方法调用。

```
 private native Class<?> defineClass0(String name, byte[] b, int off, int
len,
                                      ProtectionDomain pd);

    private native Class<?> defineClass1(String name, byte[] b, int off, int
len,
                                      ProtectionDomain pd, String
source);

    private native Class<?> defineClass2(String name, java.nio.ByteBuffer b,
                                      int off, int len, ProtectionDomain
pd,
                                      String source);
```

对应到JDK源码中分别为：

```
JNIEXPORT jclass JNICALL
Java_java_lang_ClassLoader_defineClass0(JNIEnv *env,
                                        jobject loader,
```

```
                                        jstring name,
                                        jbyteArray data,
                                        jint offset,
                                        jint length,
                                        jobject pd)

JNIEXPORT jclass JNICALL
Java_java_lang_ClassLoader_defineClass1(JNIEnv *env,
                                        jobject loader,
                                        jstring name,
                                        jbyteArray data,
                                        jint offset,
                                        jint length,
                                        jobject pd,
                                        jstring source)

JNIEXPORT jclass JNICALL
Java_java_lang_ClassLoader_defineClass2(JNIEnv *env,
                                        jobject loader,
                                        jstring name,
                                        jobject data,
                                        jint offset,
                                        jint length,
                                        jobject pd,
                                        jstring source)
```

这三个C++方法会调用到`SystemDictionary::resolve_from_stream`检查全限定名是否包含`java.`

```
klassOop SystemDictionary::resolve_from_stream(Symbol* class_name,
                                               Handle class_loader,
                                               Handle protection_domain,
                                               ClassFileStream* st,
                                               bool verify,
                                               TRAPS) {
 ...//省略无关代码，以下是并检查全限定名，若包含java.，则抛出异常。
 const char* pkg = "java/";
  if (!HAS_PENDING_EXCEPTION &&
      !class_loader.is_null() &&
      parsed_name != NULL &&
```

```
      !strncmp((const char*)parsed_name->bytes(), pkg, strlen(pkg))) {
    ResourceMark rm(THREAD);
    char* name = parsed_name->as_C_string();
    char* index = strrchr(name, '/');
    *index = '\0';
    while ((index = strchr(name, '/')) != NULL) {
      *index = '.';
    }
    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);
  }
}
```

所以破坏双亲委派真的不可重写`java.lang.String`吗?
答案：不一定，如果破坏双亲委派的时候自己将字节流转换为一个jvm可识别的class，那确实绕过`defineClass()`中的校验全限定名的逻辑，也就可以改写`java.lang.String`，并加载到JVM中，Java将不再安全。