原 【深入理解JVM】：类加载

类加载器

加载类的开放性

类加载器（ClassLoader）是Java语言的一项创 加载的第一阶段“加载”过程中，需 一个类的全限定名来获取定义此类的二进制字节流，完成这个动作的代码块就是**类加** 现的，以便让应用程序自己决定如 取所需的类。

虚拟机规范并没有指明二进制字节流要从一个(里获取、怎样获取。这种开放使得 在很多领域得到充分运用，例如：

- 从ZIP包中读取，这很常见，成为JAR，EAR，W
- 从网络中获取，最典型的应用就是Applet
- 运行时计算生成，最典型的是动态代理技术，在 ator.generateProxyClass来为特定接口生成形式为“*$Proxy”的代理类的二进制字节流
- 有其他文件生成，最典型的JSP应用，由JSP文件 ……

类加载器与类的唯一性

类加载器虽然只用于实现类的加载动作，但是 器和这个类本身共同确立其在Java虚拟机中的**唯一性**。通俗的说，JVM中两个类是否“相等”，首先就必须是同一个类加载器加载的，否则，即使这两个类来源于同一个Class文件，被同一个虚拟机加载，只要类加载器不同，那么这两个类必定是不相等的。

这里的“相等”，包括代表类的Class对象的equals()方法、isAssignableFrom()方法、isInstance()方法的返回结果，也包括使用instanceof关键字做对象所属关系判定等情况。

以下代码说明了不同的类加载器对instanceof关键字运算的结果的影响。

```java
package com.jvm.classloading;

import java.io.IOException;
import java.io.InputStream;

/**
 * 类加载器在类相等判断中的影响
 *
 * instanceof关键字
 *
 */

public class ClassLoaderTest {
    public static void main(String[] args) throws Exception {
        // 自定义类加载器
        ClassLoader myLoader = new ClassLoader() {
            @Override
            public Class<?> loadClass(String name) throws ClassNotFoundException {
                try {
                    String fileName = name.substring(name.lastIndexOf(".") + 1) + ".class";
                    InputStream is = getClass().getResourceAsStream(fileName);
                    if (is == null) {
                        return super.loadClass(fileName);
                    }
                    byte[] b = new byte[is.available()];
                    is.read(b);
                    return defineClass(name, b, 0, b.length);
                } catch (IOException e) {
                    throw new ClassNotFoundException();
                }
            }
        };

        // 使用ClassLoaderTest的类加载器加载本类
        Object obj1 = ClassLoaderTest.class.getClassLoader().loadClass("com.jvm.classloading.ClassLoaderTest").newInstance();
        System.out.println(obj1.getClass());
        System.out.println(obj1 instanceof com.jvm.classloading.ClassLoaderTest);
```

```
38
39          // 使用自定义类加载器加载本类
40          Object obj2 = myLoader.loadClass("com.jvm.classloading.ClassLoaderTest").newInstance();
41          System.out.println(obj2.getClass());
42          System.out.println(obj2 instanceof com.jvm.classloading.ClassLoaderTest);
43      }
44  }
```

输出结果：

```
1  class com.jvm.classloading.ClassLoaderTest
2  true
3  class com.jvm.classloading.ClassLoaderTest
4  false
```

myLoader是自定义的类加载器，可以用来加载与自己在同一路径下的Class文件。main函数的第一部分使用系统加载主类 .oaderTest的类加载器加载ClassLoader Test，输出显示，obj1的所属类型检查正确，这是虚拟机中有2个ClassLoaderTest类，一个是主类，另一个是main()方法中加载的类，由于这两个类使用同一个类加载器加载并且来源于同一个Class文件，因此这两个类是完全相同的。

第二部分使用自定义的类加载器加载ClassLoaderTest， `class com.jvm.classloading.ClassLoderTest` 显示，obj2确实是类 `com.jvm.classloading.ClassLoaderTest` 实例化出来的对象，但是第二句输出false。此时虚拟机中有3个ClassLoaderTest类，由于第3个类的类加载器与前面2个类加载器不同，虽然来源于同一个Class文件，但它是一个独立的类，所属类型检查是返回结果自然是false。

双亲委派模型

类加载器种类

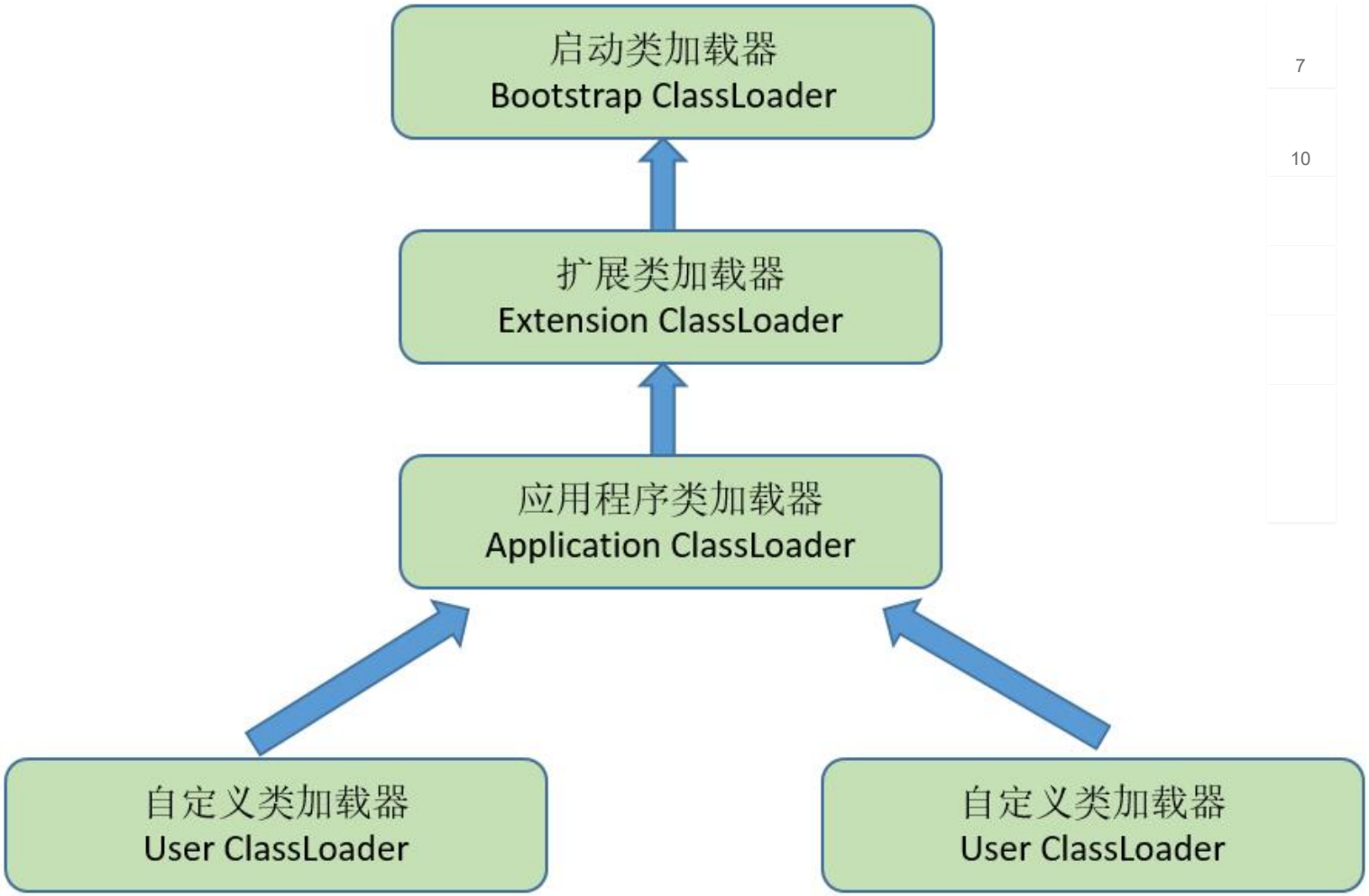
从Java虚拟机的角度来说，只存在两种不同的类加载器：一种是启动类加载器（Bootstrap ClassLoader），这个类加载器使用C++语言实现（HotSpot虚拟机中），是虚拟机自身的一部分；另一种就是所有其他的类加载器，这些类加载器都有Java语言实现，独立于虚拟机外部，并且全部继承自java.lang.ClassLoader。

从开发者的角度，类加载器可以细分为：

- 启动（Bootstrap）类加载器：负责将 Java_Home/lib下面的类库加载到内存中（比如rt.jar）。由于引导类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用，所以不允许直接通过引用进行操作。

- 标准扩展（Extension）类加载器：是由 Sun 的 ExtClassLoader（sun.misc.Launcher$ExtClassLoader）实现的。它负责将Java_Home /lib/ext或者由系统变量 java.ext.dir指定位置中的类库加载到内存中。开发者可以直接使用标准扩展类加载器。
- 应用程序（Application）类加载器：是由 Sun 的 AppClassLoader（sun.misc.Launcher$AppClassLoader）实现的。[illegible]将系统类路径（CLASSPATH）中指定的类库加载到内存中。开发者可以直接使用系统类加载器。由于这个类加载器是ClassLoader中的getSystemClassLoa[illegible]方法的返回值，因此一般称为**系统（System）加载器**。

除此之外，还有自定义的类加载器，它们之间的层次关系被称为类加载器的**双亲委派模型**。该模型要求除了顶层的启动类[illegible]外，其余的类加载器都应该有自己的父类加载器，而这种父子关系一般通过**组合（Composition）关系**来实现，而不是通过继承（Inheritance）。



7

10

双亲委派模型

双亲委派模型过程

某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。

使用双亲委派模型的好处在于**Java类随着它的类加载器一起具备了一种带有优先级的层次关系。**例如类java.lang.Object，存放在rt.jar中，无论哪一个类加载器要加载这个类，最终都是委派给处于模型最顶端的Bootstrap ClassLoader进行加载，因此Object类在程序的各种类加载器环境中都是同一个类。相反，如果没有双亲委派模型而是由各个类加载器自行加载的话，如果用户编写了一个java.lang.Object的同名类并放在ClassPath中，那系统中将会出现多个不同的Object类，程序将混乱。因此，如果开发者尝试编写一个与rt.jar类库中重名的Java类，可以正常编译，但是永远无法被加载运行。

双亲委派模型的系统实现

在java.lang.ClassLoader的loadClass()方法中，先检查是否已经被加载过，若没有加载则调用父类加载器的loadClass()方法，若父加载器为空则默认使用启动类加载器作为父加载器。如果父加载失败，则抛出ClassNotFoundException异常后，再调用自己的findClass()方法进行加载。

```
protected synchronized Class<?> loadClass(String name,boolean resolve)throws ClassNotFoundException{
    //check the class has been loaded or not
    Class c = findLoadedClass(name);
    if(c == null){
        try{
            if(parent != null){
                c = parent.loadClass(name,false);
            }else{
                c = findBootstrapClassOrNull(name);
            }
        }catch(ClassNotFoundException e){
            //if throws the exception ,the father can not complete the load
        }
        if(c == null){
            c = findClass(name);
        }
    }
    if(resolve){
        resolveClass(c);
    }
    return c;
}
```

注意，双亲委派模型是Java设计者推荐给开发者的类加载器的实现方式，并不是强制规定的。大多数的类加载器都遵循这个模型，但是JDK中也有较大规模破坏双亲模型的情况，例如线程上下文类加载器（Thread Context ClassLoader）的出现，具体分析可以参见周志明著《深入理解Java虚拟机》。

参考

1、周志明，深入理解Java虚拟机：JVM高级特性与最佳实践，机械工业出版社

2、Alexia(minmin)博客，http://www.cnblogs.com/lanxuezaipiao/p/4138511.html

7

10

双亲委派模型的理解 4157

原文地址： http://blog.csdn.net/inspiredbh/article/details/74889654 Java虚拟机先从最核心的API开始查找，防止不可信的类扮演被信...

深入理解 Tomcat（四）Tomcat 类加载器之为何违背双亲委派模型 2116

这是我们研究Tomcat的第四篇文章，前三篇文章我们搭建了源码框架，了解了tomcat的大致的设计架构， 还写了一个简单的服务器。…

双亲委派模型 1.6万

说道双亲委派模型，就要从类加载器说起。。。。。。。。。。。Java虚拟机类加载过程是把Class类文件加载到内存，并对Class文件…

双亲委派模型与类的生命周期 664

一.双亲委派模型 1.双亲委派模型是一种类加载的时候用到的一种模型，它指定了使用什么样的规则来加载类，指定了按照什么样的顺…

JVM类加载机制详解（二）类加载器与双亲委派模型 1.3万

在上一篇JVM类加载机制详解（一）JVM类加载过程中说到，类加载机制的第一个阶段加载做的工作有： 1、通过一个类的全限定名（…

Java补完之类加载机制（双亲委派模型）学习笔记 2404

前言本片文章会讨论另一个Java进阶知识要点类加载机制和双亲委派模型。概述JVM的设计团队把类加载阶段中的“通过一个类的全限…

类加载双亲委派模型 213

说到Java区别于其他语言的一大特性，自然很多人都会想到Java当初的愿景：一次编译，处处运行。而要实现这一目标自然离不开JV…

彻底弄懂Java中的equals()方法以及与"=="的区别 8504

一、问题描述： 今天在用Java实现需求的时候，发现equals()和“==”的功能傻傻分不清，导致结果产生巨大的偏差。所以，我决定花费…

相关热词 和的深入 it深入技能 docker深入 nioio深入 apache深入

个人资料

smile4lee

关注

原创	粉丝	喜欢	评论
117	**113**	**70**	**51**

等级： 博客 5　　访问： 29万+

积分： 3370　　排名： 1万+

最新文章

Java NIO 相关博文链接

Git参考文档（转载地址）

Linux Ubuntu 下安装JDK、Tomcat、Maven

Vim配置、NERDTree插件安装

Tomcat server.xml配置详解（转载地址）

7

10

个人分类

数学	1篇
操作系统	4篇
其他	2篇
数据结构	12篇
计算机网络	7篇

展开

归档

2017年7月	1篇
2016年12月	4篇
2016年9月	4篇
2016年8月	6篇
2016年6月	1篇

展开

热门文章

【深入理解JVM】：类加载器与双亲委派模型
阅读量：32509

哈希表——线性探测法、链地址法、查找成功、查找不成功的平均长度
阅读量：29043

缺页中断——FIFO、LRU、OPT这三种置换算法
阅读量：29018

计算机原码、反码、补码详解
阅读量：26668

【深入理解JVM】：Java内存模型JMM

7

10

阅读量：14927

最新评论

【深入理解JVM】：Java内存模...
henan_caiyao：不允许read和load、store和write操作之一单独出现。 这样跟直接读主内存有什么区别？...

【深入理解JVM】：OutOfMe...
LoveHaloK：[reply]LoveHaloK[/reply] 问题找到了！不劳烦博主费心回复了！

计算机原码、反码、补码详解
feng19870412：-3在内存中存储的数值是1111 1101，测试环境是 ubuntu 系统下；可以参考：https...

【深入理解JVM】：Java类继承...
limeijng：博主分享的很好，学习了。

计算机原码、反码、补码详解
huyungui1：感谢，很棒

7

10