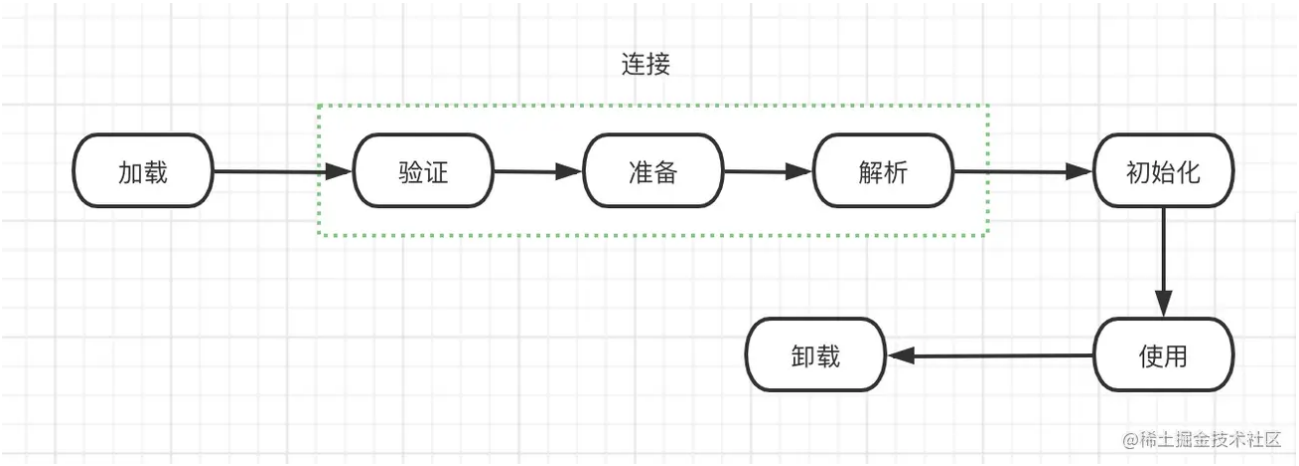
1年前 / 0评 / 35赞

[转]Android 类加载器

类的生命周期



加载阶段

加载阶段可以细分如下

- 加载类的二进制流
- 数据结构转换，将二进制流所代表的静态存储结构转化成方法区的运行时的数据结构
- 生成java.lang.Class对象，作为方法区这个类的各种数据的访问入口

加载类的二进制流的方法

1年前 / 0评 / 35赞

[转]Android 类加载器

- 运行时动态生成。我们常见的动态代理技术，在java.reflect.Proxy中就是用 ProxyGenerateProxyClass来为特定的接口生成代理的二进制流

验证

验证是连接阶段的第一步，这一阶段的目的是为了确保 Class 文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

1、文件格式验证：如是否以魔数 0xCAFEBABE 开头、主、次版本号是否在当前虚拟机处理范围之内、常量合理性验证等。
此阶段保证输入的字节流能正确地解析并存储于方法区之内，格式上符合描述一个 Java类型信息的要求。
元数据验证：是否存在父类，父类的继承链是否正确，抽象类是否实现了其父类或接口之中要求实现的所有方法，字段、方法是否与父类产生矛盾等。
2、第二阶段，保证不存在不符合 Java 语言规范的元数据信息。
3、字节码验证：通过数据流和控制流分析，确定程序语义是合法的、符合逻辑的。例如保证跳转指令不会跳转到方法体以外的字节码指令上。
4、符号引用验证：在解析阶段中发生，保证可以将符号引用转化为直接引用。

可以考虑使用 -Xverify:none 参数来关闭大部分的类验证措施，以缩短虚拟机类加载的时间。

准备

为类变量分配内存并设置类变量初始值，这些变量所使用的内存都将在方法区中进行分配。

解析

虚拟机将常量池内的符号引用替换为直接引用的过程。
解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符 7 类符号引用进行

1年前 / 0评 / 35赞

[转]Android 类加载器

到初始化阶段，才真正开始执行类中定义的 Java 程序代码，此阶段是执行 () 方法的过程。

类加载的时机

虚拟机规范规定了有且只有 5 种情况必须立即对类进行“初始化”（而加载、验证、准备自然需要在此之前开始）

- 遇到new、getstatic 和 putstatic 或 invokestatic 这4条字节码指令时，如果类没有进行过初始化，则需要先触发其初始化。对应场景是：使用 new 实例化对象、读取或设置一个类的静态字段（被 final 修饰、已在编译期把结果放入常量池的静态字段除外）、以及调用一个类的静态方法。
- 对类进行反射调用的时候，如果类没有进行过初始化，则需要先触发其初始化。
- 当初始化类的父类还没有进行过初始化，则需要先触发其父类的初始化。（而一个接口在初始化时，并不要求其父接口全部都完成了初始化）
- 虚拟机启动时，用户需要指定一个要执行的主类（包含 main() 方法的那个类），
  虚拟机会先初始化这个主类。
- 当使用 JDK 1.7 的动态语言支持时，如果一个 java.lang.invoke.MethodHandle 实例最后的解析结果 REF_getStatic、REF_putStatic、REF_invokeStatic 的方法句柄，并且这个方法句柄所对应的类没有进行过初始化，则需要先触发其初始化。

注意：

1、通过子类引用父类的静态字段，不会导致子类初始化。

2、通过数组定义来引用类，不会触发此类的初始化。MyClass[] cs = new MyClass[10];

3、常量在编译阶段会存入调用类的常量池中，本质上并没有直接引用到定义常量的类，因此不会触发定义常量的类的初始化。

类加载器

1年前 / 0评 / 35赞

[转]Android 类加载器

将 class 文件二进制数据放入方法区内，然后在堆内（heap）创建一个 java.lang.Class 对象，Class 对象封装了类在方法区内的数据结构，并且向开发者提供了访问方法区内的数据结构的接口。

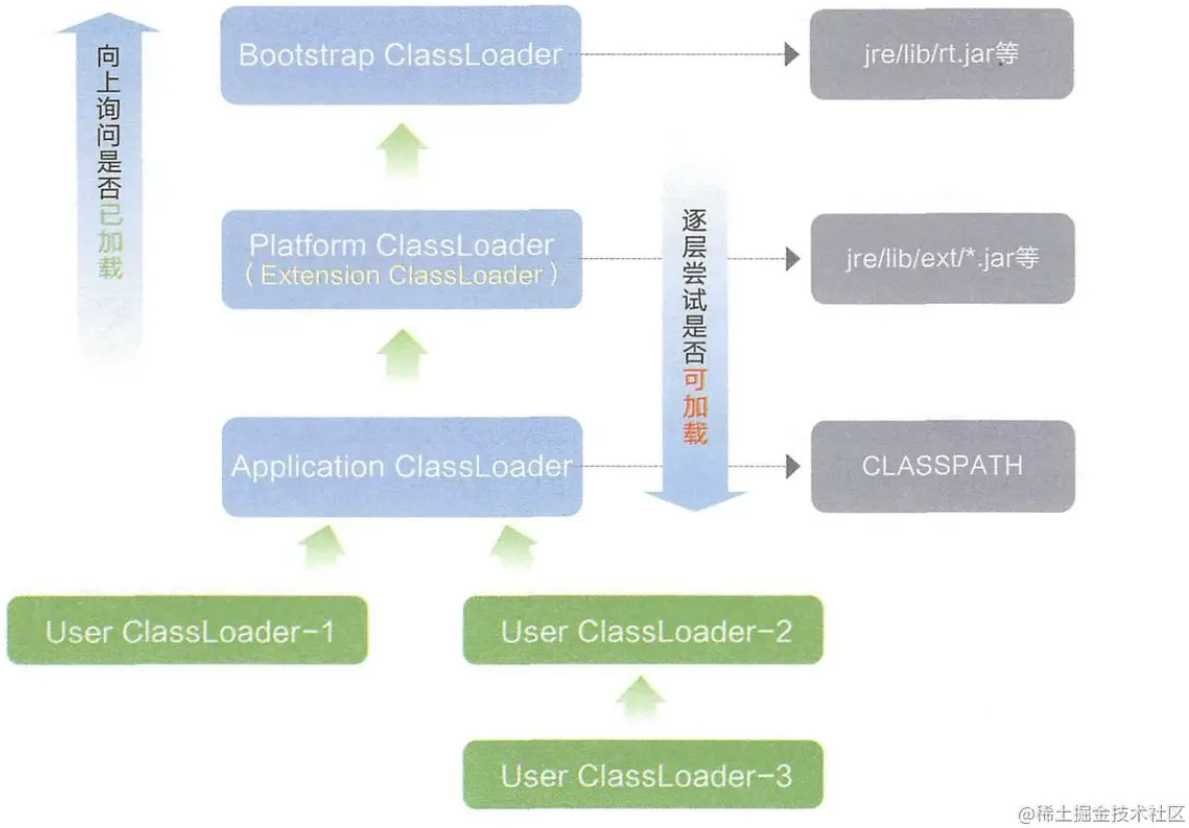
类的唯一性

对于任意一个类，都需要由加载它的类加载器和这个类本身一同确立其在Java虚拟机中的唯一性。

即使两个类来源于同一个 Class 文件，被同一个虚拟机加载，只要加载它们的类加载器不同，那这两个类也不相等。

这里所指的“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

双亲委托机制



1年前 / 0评 / 35赞

[转]Android 类加载器

应该传送到顶层的启动类加载器中，只有当父加载器反馈自己无法完成这个加载请求（它的搜索范围中没有找到所需的类）时，子加载器才会尝试自己去加载。

```
    protected Class<?> loadClass(String name, boolean resolve)
        throws ClassNotFoundException
    {
            // First, check if the class has already been loaded
//先从缓存中加没加载这个类
            Class<?> c = findLoadedClass(name);
            if (c == null) {
                try {
                    if (parent != null) {
//从parent中加载
                        c = parent.loadClass(name, false);
                    } else {
                        c = findBootstrapClassOrNull(name);
                    }
                } catch (ClassNotFoundException e) {
                    // ClassNotFoundException thrown if class not found
                    // from the non-null parent class loader
                }
//加载不到，就自己加载
                if (c == null) {
                    // If still not found, then invoke findClass in order
                    // to find the class.
                    c = findClass(name);
                }
            }
            return c;
    }
```

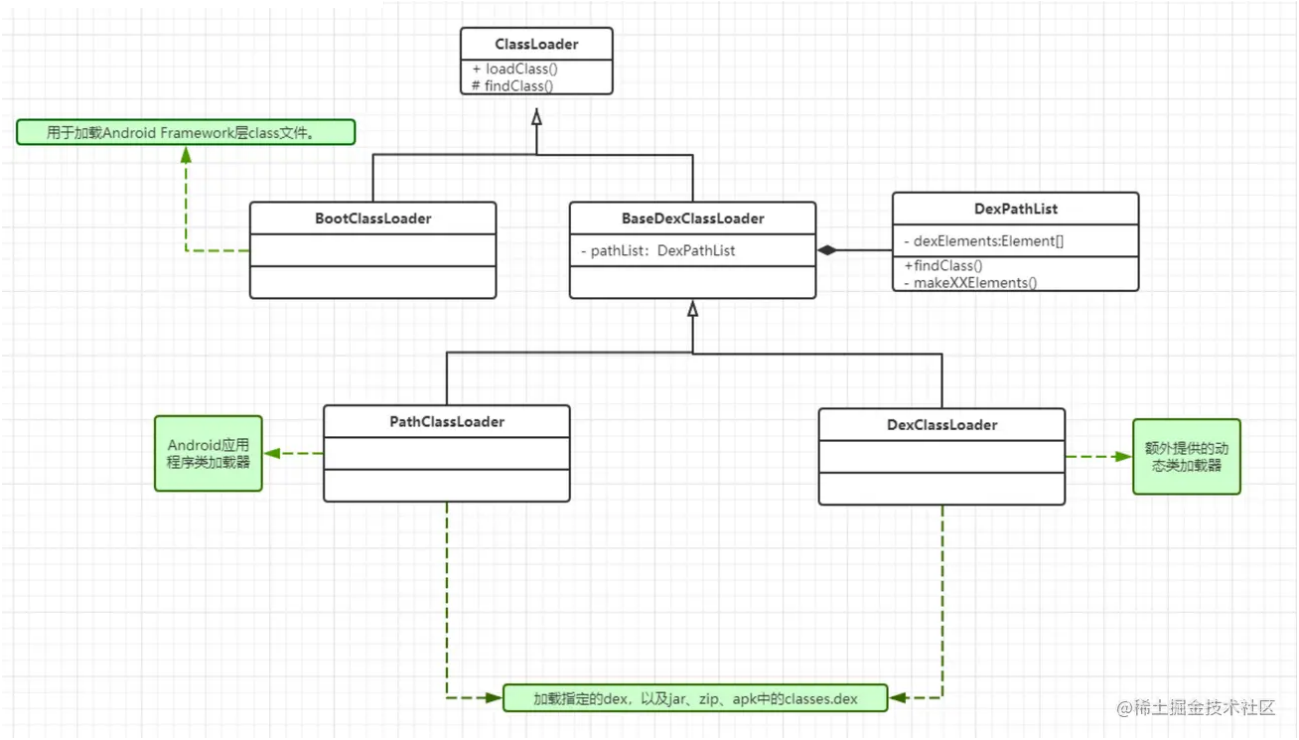
好处

避免重复加载，当父加载器已经加载了该类的时候，就没有必要子ClassLoader再加载一次。

安全性考虑，防止核心API库被随意篡改。

1年前 / 0评 / 35赞

[转]Android 类加载器



- ClassLoader是一个抽象类，定义了ClassLoader的主要功能
- BootClassLoader是ClassLoader的子类（注意不是内部类，有些材料上说是内部类，是不对的），用于加载一些系统Framework层级需要的类，是Android平台上所有的ClassLoader的最终parent
- SecureClassLoader扩展了ClassLoader类，加入了权限方面的功能，加强了安全性
- URLClassLoader继承SecureClassLoader，用来通过URI路径从jar文件和文件夹中加载类和资源，在Android中基本无法使用
- BaseDexClassLoader是实现了Android ClassLoader的大部分功能
- PathClassLoader加载应用程序的类，会加载/data/app目录下的dex文件以及包含dex的apk文件或者java文件（有些材料上说他也会加载系统类，我没有找到，这里存疑）
- DexClassLoader可以加载自定义dex文件以及包含dex的apk文件或jar文件，支持从SD卡进行加载。我们使用插件化技术的时候会用到
- InMemoryDexClassLoader用于加载内存中的dex文件

1年前 / 0评 / 35赞

[转]Android 类加载器

-> ClassLoader.java 类

```
protected Class<?> loadClass(String name, boolean resolve)   throws
ClassNotFoundException
{
        // First, check if the class has already been loaded
        Class<?> c = findLoadedClass(name);
        if (c == null) {
            try {
                if (parent != null) {
                    c = parent.loadClass(name, false);
                } else {
                    c = findBootstrapClassOrNull(name);
                }
            } catch (ClassNotFoundException e) {
                // ClassNotFoundException thrown if class not found
                // from the non-null parent class loader
            }

            if (c == null) {
                // If still not found, then invoke findClass in order
                // to find the class.
                c = findClass(name);
            }
        }
        return c;
}
```

findClass方法由子类实现

```
protected Class<?> findClass(String name) throws ClassNotFoundException {
    throw new ClassNotFoundException(name);
}
```

BaseDexClassLoader类中findClass方法

1年前 / 0评 / 35赞

[转]Android 类加载器

```
                    ppressedExceptions = new ArrayList<Throwable>();
    // pathList是DexPathList，是具体存放代码的地方。
    Class c = pathList.findClass(name, suppressedExceptions);
    if (c == null) {
        ClassNotFoundException cnfe = new ClassNotFoundException(
                "Didn't find class "" + name + "" on path: " + pathList);
        for (Throwable t : suppressedExceptions) {
            cnfe.addSuppressed(t);
        }
        throw cnfe;
    }
    return c;
}
```

DexPathList的findClass方法

```
public Class<?> findClass(String name, List<Throwable> suppressed) {
    for (Element element : dexElements) {
        Class<?> clazz = element.findClass(name, definingContext, suppressed);
        if (clazz != null) {
            return clazz;
        }
    }

    if (dexElementsSuppressedExceptions != null) {
        suppressed.addAll(Arrays.asList(dexElementsSuppressedExceptions));
    }
    return null;
}
```

Element的findClass方法

```
public Class<?> findClass(String name, List<Throwable> suppressed) {
    for (Element element : dexElements) {
        Class<?> clazz = element.findClass(name, definingContext, suppressed);
        if (clazz != null) {
            return clazz;
        }
```

1年前 / 0评 / 35赞

[转]Android 类加载器

```
        if (dexElementsSuppressedExceptions != null) {
            suppressed.addAll(Arrays.asList(dexElementsSuppressedExceptions));
        }
        return null;
    }
```

```
    public Class<?> findClass(String name, ClassLoader definingContext,
            List<Throwable> suppressed) {
        return dexFile != null ? dexFile.loadClassBinaryName(name, definingContext,
    suppressed) : null;
    }
```

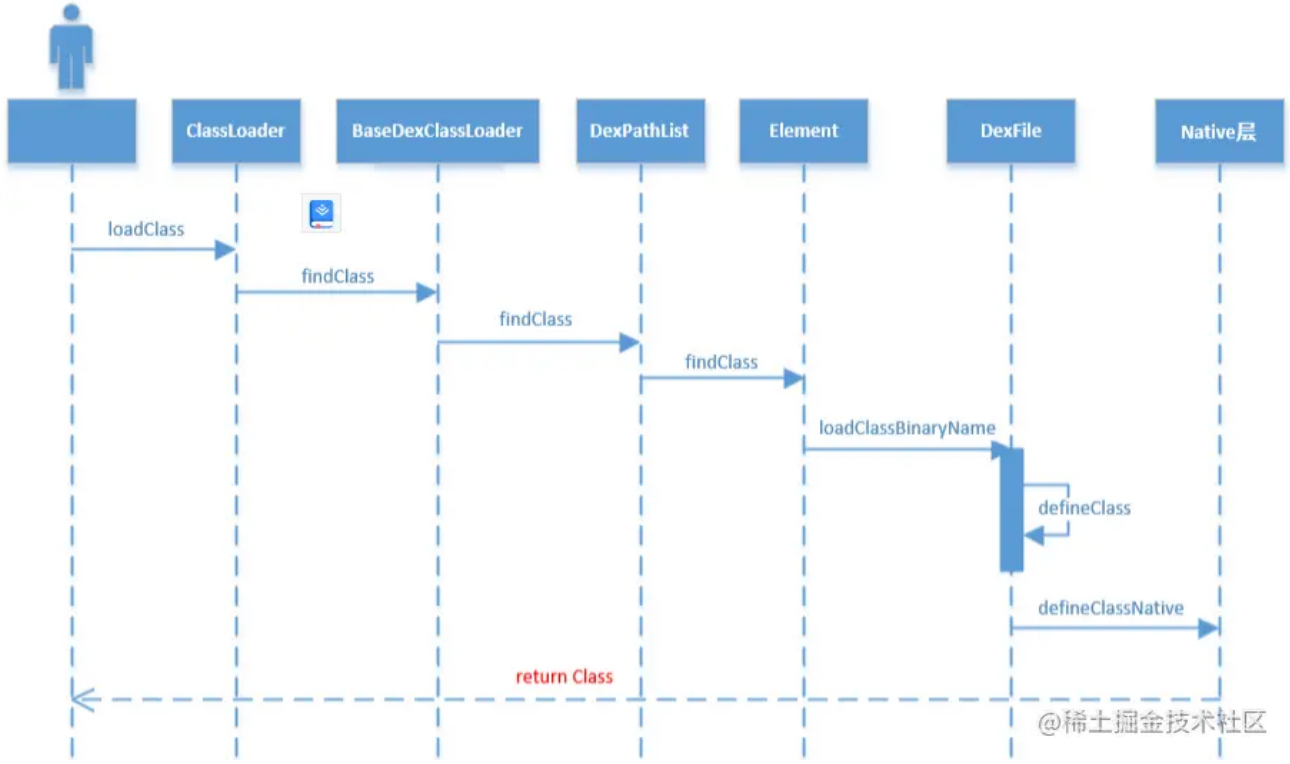
```
    public Class loadClassBinaryName(String name, ClassLoader loader,
    List<Throwable> suppressed) {
        return defineClass(name, loader, mCookie, this, suppressed);
    }

    private static Class defineClass(String name, ClassLoader loader, Object
    cookie,
                                     DexFile dexFile, List<Throwable> suppressed) {
        Class result = null;
        try {
            result = defineClassNative(name, loader, cookie, dexFile);
        } catch (NoClassDefFoundError e) {
            if (suppressed != null) {
                suppressed.add(e);
            }
        } catch (ClassNotFoundException e) {
            if (suppressed != null) {
                suppressed.add(e);
            }
        }
        return result;
    }

    // 调用 Native 层代码
```

[转]Android 类加载器

1年前 / 0评 / 35赞



作者：Arrom

链接：https://juejin.cn/post/7038476576366788621

来源：稀土掘金

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

内功　　好文要转

« Android虚拟机　　Android换肤方案解析 »

发表回复

显示名称	电子邮箱地址	网站地址

[转]Android 类加载器

1年前 / 0评 / 35赞

发表评论

推荐资源 留言板

© 2023 Alias's blog Theme by Adams