**[DexClassLoader和PathClassLoader的区别](http://blog.csdn.net/mynameishuangshuai/article/details/52737581)**

在使用**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/mynameishuangshuai/article/details/_blank)**虚拟机时，我们经常自定义继承自ClassLoader的类加载器。然后通过defineClass方法来从一个二进制流中加载Class。而在**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/mynameishuangshuai/article/details/_blank)**中我们无法这么使用，Android中ClassLoader的defineClass方法具体是调用VMClassLoader的defineClass本地静态方法。而这个本地方法什么都没做，只是抛出了一个“UnsupportedOperationException”异常。   
    既然在Dalvik虚拟机里，ClassLoader不好用，那么Android官方为了解决这个问题，帮我们从ClassLoader中派生出了两个类：**DexClassLoader**和**PathClassLoader**。咋一看两者很像，那么究竟二者在使用上面有何不同，这里我和大家一起探讨一下。

首先来看一下二者的构造方法

### DexClassLoader

public DexClassLoader (String dexPath, String dexOutputDir, String libPath, ClassLoader parent)

参数详解：

dexPath：dex文件路径列表，多个路径使用”:”分隔   
dexOutputDir：经过优化的dex文件（odex）文件输出目录   
libPath：动态库路径（将被添加到app动态库搜索路径列表中）   
parent：这是一个ClassLoader，这个参数的主要作用是保留java中ClassLoader的委托机制（优先父类加载器加载classes，由上而下的加载机制，防止重复加载类字节码）

DexClassLoader是一个可以从包含classes.dex实体的.jar或.apk文件中加载classes的类加载器。可以用于实现dex的动态加载、代码热更新等等。这个类加载器必须要一个app的私有、可写目录来缓存经过优化的classes（odex文件），使用Context.getDir(String, int)方法可以创建一个这样的目录，例如：

File dexOutputDir = context.getDir(“dex”, 0);

### PathClassLoader

PathClassLoader提供两个常用构造方法

public PathClassLoader (String path, ClassLoader parent)

public PathClassLoader (String path, String libPath, ClassLoader parent)

参数详解：

path：文件或者目录的列表   
libPath：包含lib库的目录列表   
parent：父类加载器

PathClassLoader提供一个简单的ClassLoader实现，可以操作在本地文件系统的文件列表或目录中的classes，但不可以从网络中加载classes。

为了便于理解，我们查看一下二者的源码：



// DexClassLoader.java

public class DexClassLoader extends BaseDexClassLoader {

public DexClassLoader(String dexPath, String optimizedDirectory,

String libraryPath, ClassLoader parent) {

super(dexPath, new File(optimizedDirectory), libraryPath, parent);

}

}

// PathClassLoader.java

public class PathClassLoader extends BaseDexClassLoader {

public PathClassLoader(String dexPath, ClassLoader parent) {

super(dexPath, null, null, parent);

}

public PathClassLoader(String dexPath, String libraryPath,

ClassLoader parent) {

super(dexPath, null, libraryPath, parent);

}

}

很明显两者都继承于BaseDexClassLoader类，并做了一下封装，具体的实现还是在父类里。不难看出，主要的区别在于PathClassLoader的optimizedDirectory参数只能是null，那么optimizedDirectory是做什么用的呢？我们进BaseDexClassLoader去看看这个参数。

public BaseDexClassLoader(String dexPath, File optimizedDirectory,

String libraryPath, ClassLoader parent) {

super(parent);

this.originalPath = dexPath;

this.pathList = new DexPathList(this, dexPath, libraryPath, optimizedDirectory);

}

代码中与optimizedDirectory有关的地方是new 一个DexPathList实例。

public DexPathList(ClassLoader definingContext, String dexPath,

String libraryPath, File optimizedDirectory) {

……

this.dexElements = makeDexElements(splitDexPath(dexPath), optimizedDirectory);

}

private static Element[] makeDexElements(ArrayList<File> files,

File optimizedDirectory) {

ArrayList<Element> elements = new ArrayList<Element>();

for (File file : files) {

ZipFile zip = null;

DexFile dex = null;

String name = file.getName();

if (name.endsWith(DEX\_SUFFIX)) {

dex = loadDexFile(file, optimizedDirectory);

} else if (name.endsWith(APK\_SUFFIX) || name.endsWith(JAR\_SUFFIX)

|| name.endsWith(ZIP\_SUFFIX)) {

zip = new ZipFile(file);

}

……

if ((zip != null) || (dex != null)) {

elements.add(new Element(file, zip, dex));

}

}

return elements.toArray(new Element[elements.size()]);

}

private static DexFile loadDexFile(File file, File optimizedDirectory)

throws IOException {

if (optimizedDirectory == null) {

return new DexFile(file);

} else {

String optimizedPath = optimizedPathFor(file, optimizedDirectory);

return DexFile.loadDex(file.getPath(), optimizedPath, 0);

}

}

/\*\*

\* Converts a dex/jar file path and an output directory to an

\* output file path for an associated optimized dex file.

\*/

private static String optimizedPathFor(File path,

File optimizedDirectory) {

String fileName = path.getName();

if (!fileName.endsWith(DEX\_SUFFIX)) {

int lastDot = fileName.lastIndexOf(".");

if (lastDot < 0) {

fileName += DEX\_SUFFIX;

} else {

StringBuilder sb = new StringBuilder(lastDot + 4);

sb.append(fileName, 0, lastDot);

sb.append(DEX\_SUFFIX);

fileName = sb.toString();

}

}

File result = new File(optimizedDirectory, fileName);

return result.getPath();

}

optimizedDirectory是用来缓存我们需要加载的dex文件的，并创建一个DexFile对象，如果它为null，那么会直接使用dex文件原有的路径来创建DexFile对象。

optimizedDirectory必须是一个内部存储路径，无论哪种动态加载，加载的可执行文件一定要存放在内部存储。DexClassLoader可以指定自己的optimizedDirectory，所以它可以加载外部的dex，因为这个dex会被复制到内部路径的optimizedDirectory；而PathClassLoader没有optimizedDirectory，所以它只能加载内部的dex，这些大都是存在系统中已经安装过的apk里面的。

### 通过以上的分析，我们可以得出二者功能上的区别

DexClassLoader：能够加载未安装的jar/apk/dex   
PathClassLoader：只能加载系统中已经安装过的apk