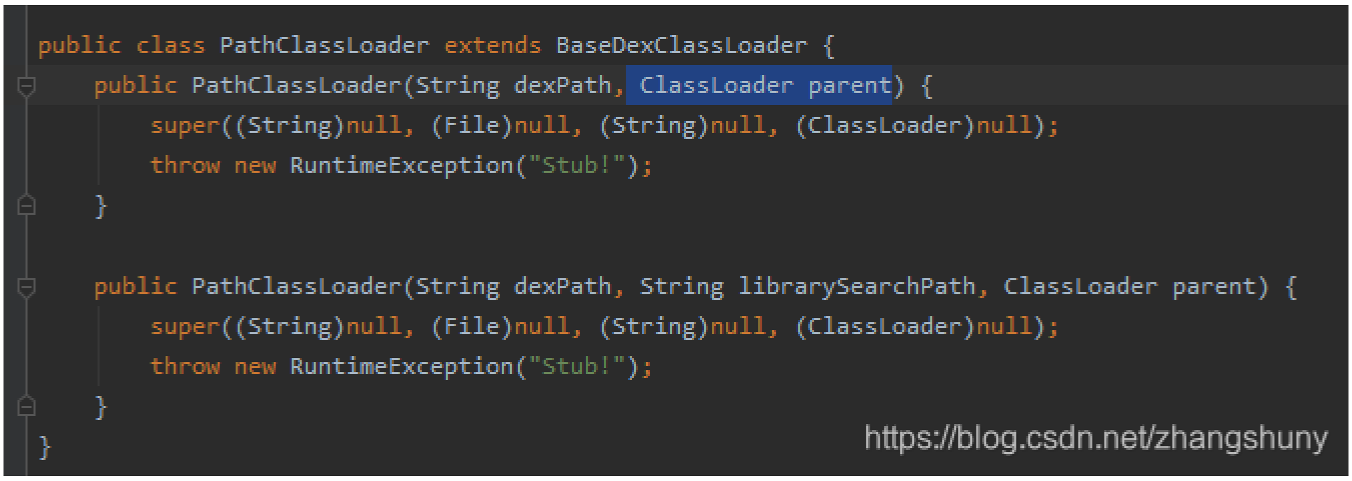
目录

写在前面

双亲委托机制中，当加载一个Class的时候，如果当前ClassLoader有父加载器的时候用父加载器加载。为什么PathClassLoader的父加载器(parent)是BootClassLoader?

查看PathClassLoader的源码

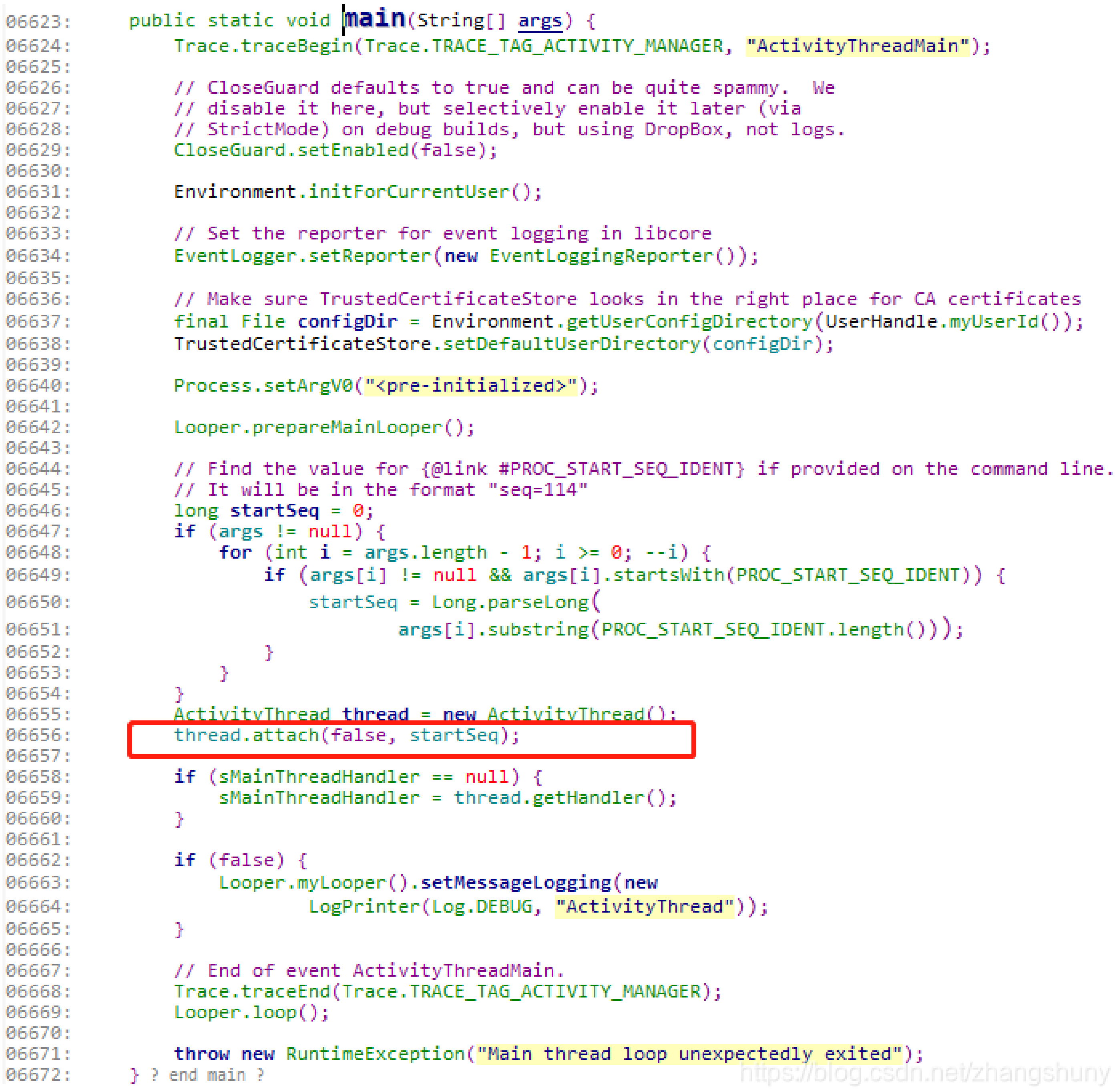


```java
public class PathClassLoader extends BaseDexClassLoader {
    public PathClassLoader(String dexPath, ClassLoader parent) {
        super((String)null, (File)null, (String)null, (ClassLoader)null);
        throw new RuntimeException("Stub!");
    }

    public PathClassLoader(String dexPath, String librarySearchPath, ClassLoader parent) {
        super((String)null, (File)null, (String)null, (ClassLoader)null);
        throw new RuntimeException("Stub!");
    }
}
```

从上图可知，PathClassLoader构造函数传入了一个ClassLoader类型的参数parent。

PathClassLoader是何时创建以及parent是什么
从哪看呢？由于ActivityThread是我们常说的UI线程，ActivityThread类中的main()方法是整个应用的入口。所以我们来看一下ActivityThread的main()。

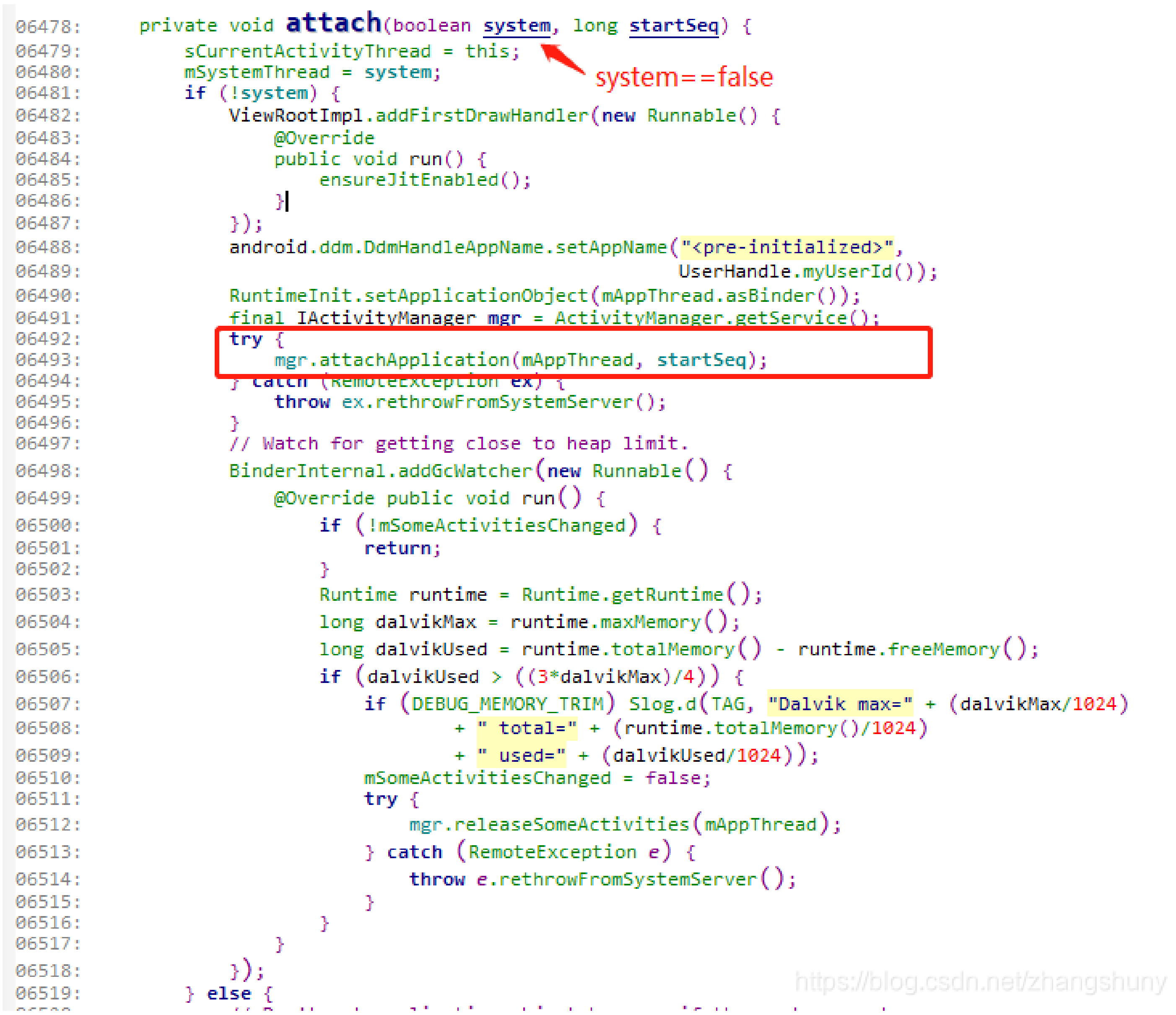
ActivityThread类的main方法



```
public static void main(String[] args) {
    Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER, "ActivityThreadMain");

    // CloseGuard defaults to true and can be quite spammy.  We
    // disable it here, but selectively enable it later (via
    // StrictMode) on debug builds, but using DropBox, not logs.
    CloseGuard.setEnabled(false);

    Environment.initForCurrentUser();

    // Set the reporter for event logging in libcore
    EventLogger.setReporter(new EventLoggingReporter());

    // Make sure TrustedCertificateStore looks in the right place for CA certificates
    final File configDir = Environment.getUserConfigDirectory(UserHandle.myUserId());
    TrustedCertificateStore.setDefaultUserDirectory(configDir);

    Process.setArgV0("<pre-initialized>");

    Looper.prepareMainLooper();

    // Find the value for {@link #PROC_START_SEQ_IDENT} if provided on the command line.
    // It will be in the format "seq=114"
    long startSeq = 0;
    if (args != null) {
        for (int i = args.length - 1; i >= 0; --i) {
            if (args[i] != null && args[i].startsWith(PROC_START_SEQ_IDENT)) {
                startSeq = Long.parseLong(
                        args[i].substring(PROC_START_SEQ_IDENT.length()));
            }
        }
    }
    ActivityThread thread = new ActivityThread();
    thread.attach(false, startSeq);

    if (sMainThreadHandler == null) {
        sMainThreadHandler = thread.getHandler();
    }

    if (false) {
        Looper.myLooper().setMessageLogging(new
                LogPrinter(Log.DEBUG, "ActivityThread"));
    }

    // End of event ActivityThreadMain.
    Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
    Looper.loop();

    throw new RuntimeException("Main thread loop unexpectedly exited");
} ? end main ?
```

main方法中进行了MainLooper的初始化和主线程的创建以及主线程Handler的初始化

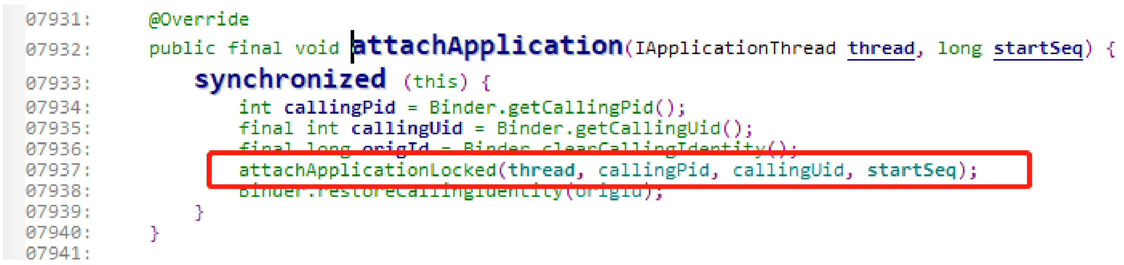
这里我们关注一下attach方法



```java
    private void attach(boolean system, long startSeq) {
        sCurrentActivityThread = this;
        mSystemThread = system;
        if (!system) {
            ViewRootImpl.addFirstDrawHandler(new Runnable() {
                @Override
                public void run() {
                    ensureJitEnabled();
                }
            });
            android.ddm.DdmHandleAppName.setAppName("<pre-initialized>",
                                                    UserHandle.myUserId());
            RuntimeInit.setApplicationObject(mAppThread.asBinder());
            final IActivityManager mgr = ActivityManager.getService();
            try {
                mgr.attachApplication(mAppThread, startSeq);
            } catch (RemoteException ex) {
                throw ex.rethrowFromSystemServer();
            }
            // Watch for getting close to heap limit.
            BinderInternal.addGcWatcher(new Runnable() {
                @Override public void run() {
                    if (!mSomeActivitiesChanged) {
                        return;
                    }
                    Runtime runtime = Runtime.getRuntime();
                    long dalvikMax = runtime.maxMemory();
                    long dalvikUsed = runtime.totalMemory() - runtime.freeMemory();
                    if (dalvikUsed > ((3*dalvikMax)/4)) {
                        if (DEBUG_MEMORY_TRIM) Slog.d(TAG, "Dalvik max=" + (dalvikMax/1024)
                                + " total=" + (runtime.totalMemory()/1024)
                                + " used=" + (dalvikUsed/1024));
                        mSomeActivitiesChanged = false;
                        try {
                            mgr.releaseSomeActivities(mAppThread);
                        } catch (RemoteException e) {
                            throw e.rethrowFromSystemServer();
                        }
                    }
                }
            });
        } else {
```

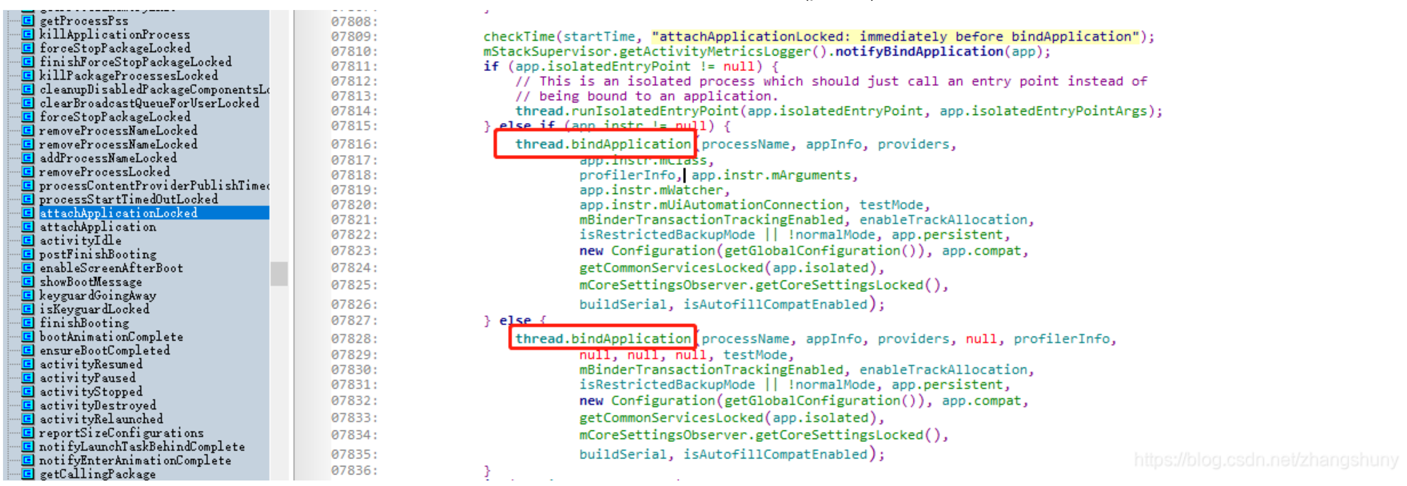
在ActivityThread的attach方法中，ActivityManagerService通过attachApplication方法,将ApplicationThread对象绑定到ActivityManagerService上，mAppThread对象所对应的类是ApplicatinThread，是ActivityThread的内部类，实现了IBinder接口，用于ActivityThread和ActivityManagerService的进程间通信

我们接着看attachApplication方法



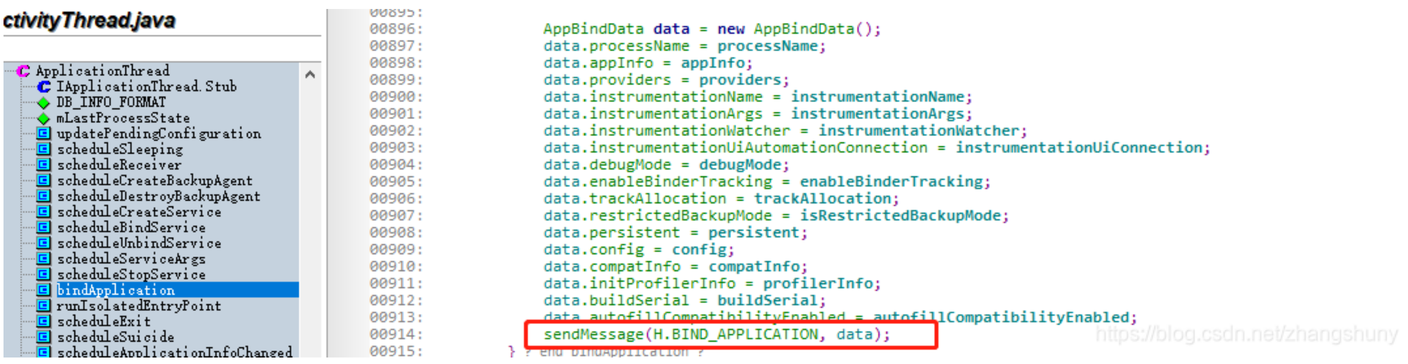
```java
    @Override
    public final void attachApplication(IApplicationThread thread, long startSeq) {
        synchronized (this) {
            int callingPid = Binder.getCallingPid();
            final int callingUid = Binder.getCallingUid();
            final long origId = Binder.clearCallingIdentity();
            attachApplicationLocked(thread, callingPid, callingUid, startSeq);
            Binder.restoreCallingIdentity(origId);
        }
    }
```

方法中调用了attachApplicationLocked方法



```
            checkTime(startTime, "attachApplicationLocked: immediately before bindApplication");
            mStackSupervisor.getActivityMetricsLogger().notifyBindApplication(app);
            if (app.isolatedEntryPoint != null) {
                // This is an isolated process which should just call an entry point instead of
                // being bound to an application.
                thread.runIsolatedEntryPoint(app.isolatedEntryPoint, app.isolatedEntryPointArgs);
            } else if (app.instr != null) {
                thread.bindApplication(processName, appInfo, providers,
                        app.instr.mClass,
                        profilerInfo, app.instr.mArguments,
                        app.instr.mWatcher,
                        app.instr.mUiAutomationConnection, testMode,
                        mBinderTransactionTrackingEnabled, enableTrackAllocation,
                        isRestrictedBackupMode || !normalMode, app.persistent,
                        new Configuration(getGlobalConfiguration()), app.compat,
                        getCommonServicesLocked(app.isolated),
                        mCoreSettingsObserver.getCoreSettingsLocked(),
                        buildSerial, isAutofillCompatEnabled);
            } else {
                thread.bindApplication(processName, appInfo, providers, null, profilerInfo,
                        null, null, null, testMode,
                        mBinderTransactionTrackingEnabled, enableTrackAllocation,
                        isRestrictedBackupMode || !normalMode, app.persistent,
                        new Configuration(getGlobalConfiguration()), app.compat,
                        getCommonServicesLocked(app.isolated),
                        mCoreSettingsObserver.getCoreSettingsLocked(),
                        buildSerial, isAutofillCompatEnabled);
            }
```

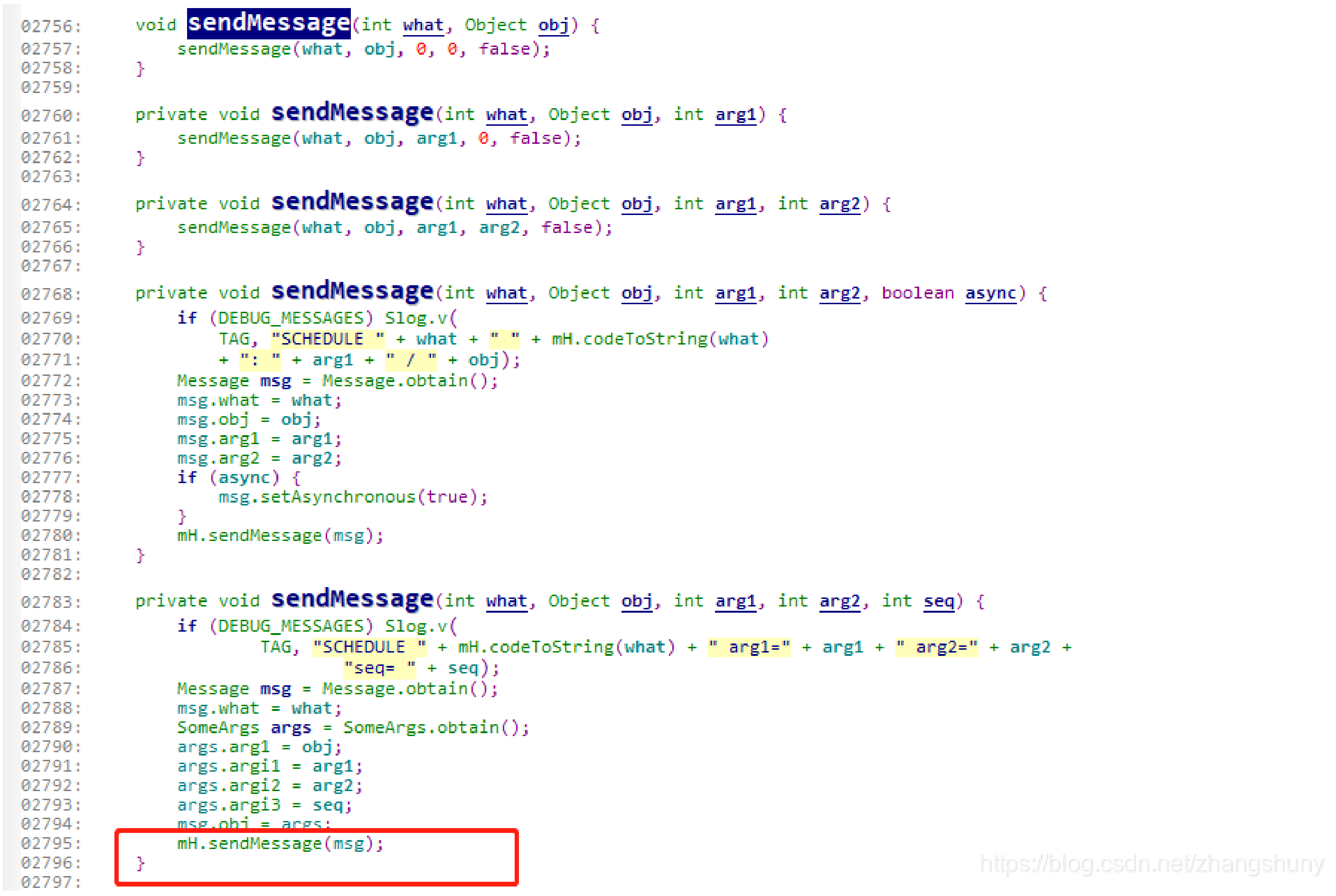
attachApplicationLocked方法中调用了mAppThread的bindApplication方法



```
            AppBindData data = new AppBindData();
            data.processName = processName;
            data.appInfo = appInfo;
            data.providers = providers;
            data.instrumentationName = instrumentationName;
            data.instrumentationArgs = instrumentationArgs;
            data.instrumentationWatcher = instrumentationWatcher;
            data.instrumentationUiAutomationConnection = instrumentationUiConnection;
            data.debugMode = debugMode;
            data.enableBinderTracking = enableBinderTracking;
            data.trackAllocation = trackAllocation;
            data.restrictedBackupMode = isRestrictedBackupMode;
            data.persistent = persistent;
            data.config = config;
            data.compatInfo = compatInfo;
            data.initProfilerInfo = profilerInfo;
            data.buildSerial = buildSerial;
            data.autofillCompatibilityEnabled = autofillCompatibilityEnabled;
            sendMessage(H.BIND_APPLICATION, data);
        } // end bindApplication
```

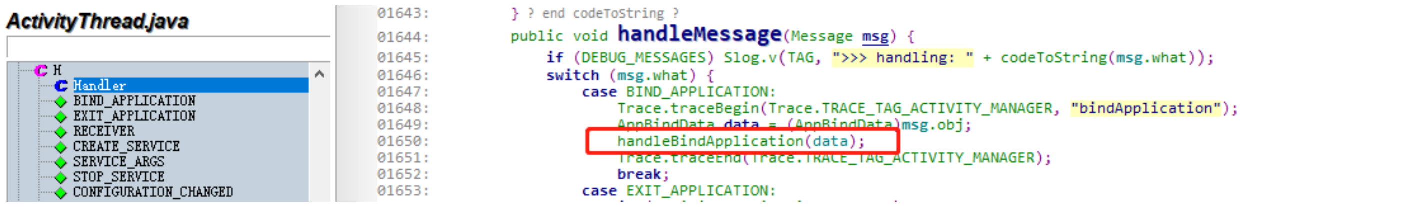
bindApplication方法中创建了AppBindData对象，对象中设置了线程信息、application信息等

然后调用了sendMessage发消息，what = BIND_APPLICATION



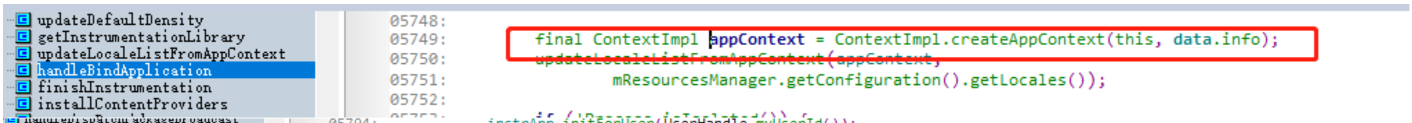
```
    void sendMessage(int what, Object obj) {
        sendMessage(what, obj, 0, 0, false);
    }

    private void sendMessage(int what, Object obj, int arg1) {
        sendMessage(what, obj, arg1, 0, false);
    }

    private void sendMessage(int what, Object obj, int arg1, int arg2) {
        sendMessage(what, obj, arg1, arg2, false);
    }

    private void sendMessage(int what, Object obj, int arg1, int arg2, boolean async) {
        if (DEBUG_MESSAGES) Slog.v(
            TAG, "SCHEDULE " + what + " " + mH.codeToString(what)
            + ": " + arg1 + " / " + obj);
        Message msg = Message.obtain();
        msg.what = what;
        msg.obj = obj;
        msg.arg1 = arg1;
        msg.arg2 = arg2;
        if (async) {
            msg.setAsynchronous(true);
        }
        mH.sendMessage(msg);
    }

    private void sendMessage(int what, Object obj, int arg1, int arg2, int seq) {
        if (DEBUG_MESSAGES) Slog.v(
                TAG, "SCHEDULE " + mH.codeToString(what) + " arg1=" + arg1 + " arg2=" + arg2 +
                    "seq= " + seq);
        Message msg = Message.obtain();
        msg.what = what;
        SomeArgs args = SomeArgs.obtain();
        args.arg1 = obj;
        args.argi1 = arg1;
        args.argi2 = arg2;
        args.argi3 = seq;
        msg.obj = args;
        mH.sendMessage(msg);
    }
```

sendMessage中封装了一下给主线程handler：mH 发消息

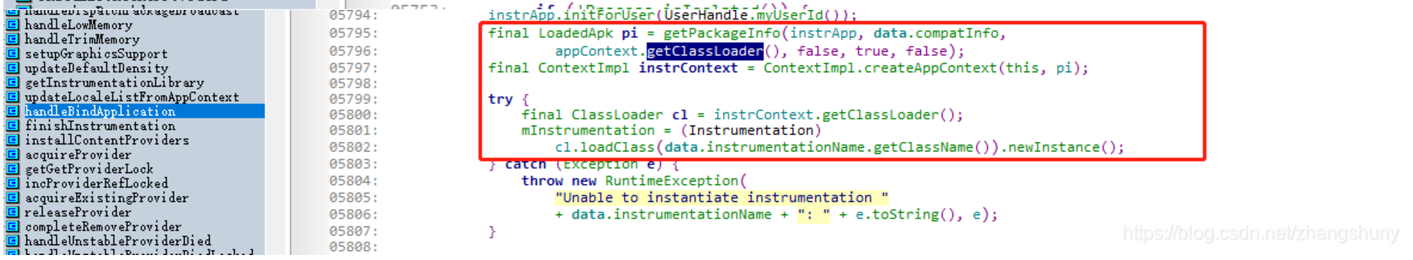


```
    } ? end codeToString ?
    public void handleMessage(Message msg) {
        if (DEBUG_MESSAGES) Slog.v(TAG, ">>> handling: " + codeToString(msg.what));
        switch (msg.what) {
            case BIND_APPLICATION:
                Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER, "bindApplication");
                AppBindData data = (AppBindData)msg.obj;
                handleBindApplication(data);
                Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
                break;
            case EXIT_APPLICATION:
```

主线程handler的handleMessage方法中调用handleBindApplication方法



```
        final ContextImpl appContext = ContextImpl.createAppContext(this, data.info);
        ~~updateLocaleListFromAppContext(appContext,~~
                mResourcesManager.getConfiguration().getLocales());
```



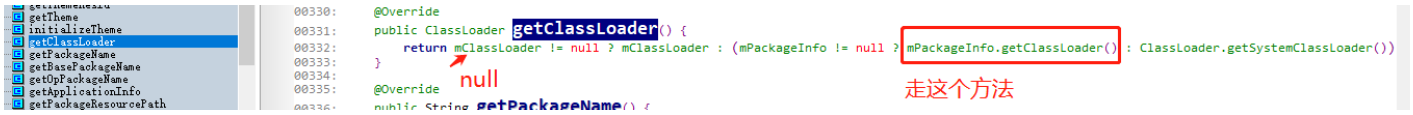
```
            instrApp.initForUser(UserHandle.myUserId());
            final LoadedApk pi = getPackageInfo(instrApp, data.compatInfo,
                    appContext.getClassLoader(), false, true, false);
            final ContextImpl instrContext = ContextImpl.createAppContext(this, pi);

            try {
                final ClassLoader cl = instrContext.getClassLoader();
                mInstrumentation = (Instrumentation)
                    cl.loadClass(data.instrumentationName.getClassName()).newInstance();
            } catch (Exception e) {
                throw new RuntimeException(
                    "Unable to instantiate instrumentation "
                    + data.instrumentationName + ": " + e.toString(), e);
            }
```

handleBindApplication方法中代码比较多，先关注两点：

1.创建了ContextImpl

2.调用了ContextImpl的getClassLoader()方法

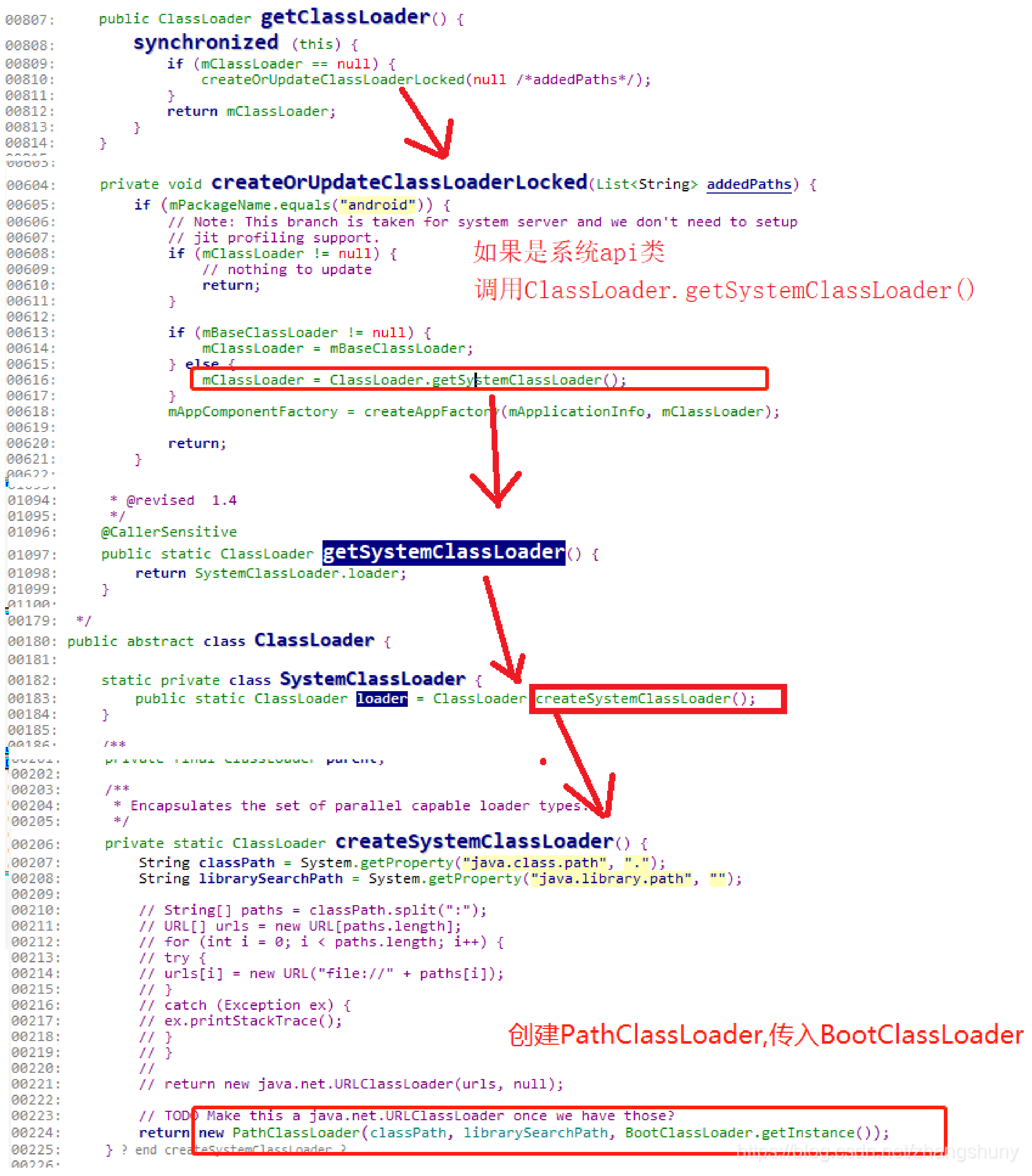
看一下如何getClassLoader的



```
    @Override
    public ClassLoader getClassLoader() {
        return mClassLoader != null ? mClassLoader : (mPackageInfo != null ? mPackageInfo.getClassLoader() : ClassLoader.getSystemClassLoader())
    }

    @Override
    public String getPackageName() {
```

mClassLoader和mPackageInfo是构造函数中传入的，mClassLoader传入的是null，mPackageInfo

传入的是data.info所以调用mPackageInfo.getClassLoader()



```
    public ClassLoader getClassLoader() {
        synchronized (this) {
            if (mClassLoader == null) {
                createOrUpdateClassLoaderLocked(null /*addedPaths*/);
            }
            return mClassLoader;
        }
    }

    private void createOrUpdateClassLoaderLocked(List<String> addedPaths) {
        if (mPackageName.equals("android")) {
            // Note: This branch is taken for system server and we don't need to setup
            // jit profiling support.
            if (mClassLoader != null) {
                // nothing to update
                return;
            }

            if (mBaseClassLoader != null) {
                mClassLoader = mBaseClassLoader;
            } else {
                mClassLoader = ClassLoader.getSystemClassLoader();
            }
            mAppComponentFactory = createAppFactory(mApplicationInfo, mClassLoader);

            return;
        }
```

如果是系统api类

调用ClassLoader.getSystemClassLoader()

```
     * @revised  1.4
     */
    @CallerSensitive
    public static ClassLoader getSystemClassLoader() {
        return SystemClassLoader.loader;
    }
```

```
 */
public abstract class ClassLoader {

    static private class SystemClassLoader {
        public static ClassLoader loader = ClassLoader.createSystemClassLoader();
    }

    /**
    private final ClassLoader parent;

    /**
     * Encapsulates the set of parallel capable loader types.
     */
    private static ClassLoader createSystemClassLoader() {
        String classPath = System.getProperty("java.class.path", ".");
        String librarySearchPath = System.getProperty("java.library.path", "");

        // String[] paths = classPath.split(":");
        // URL[] urls = new URL[paths.length];
        // for (int i = 0; i < paths.length; i++) {
        // try {
        // urls[i] = new URL("file://" + paths[i]);
        // }
        // catch (Exception ex) {
        // ex.printStackTrace();
        // }
        // }
        //
        // return new java.net.URLClassLoader(urls, null);

        // TODO Make this a java.net.URLClassLoader once we have those?
        return new PathClassLoader(classPath, librarySearchPath, BootClassLoader.getInstance());
    } ? end createSystemClassLoader ?
```

创建PathClassLoader,传入BootClassLoader

总结

我们写的应用程序的Class使用PathClassLoader加载，PathClassLoader的父加载器(parent)是BootClassLoader