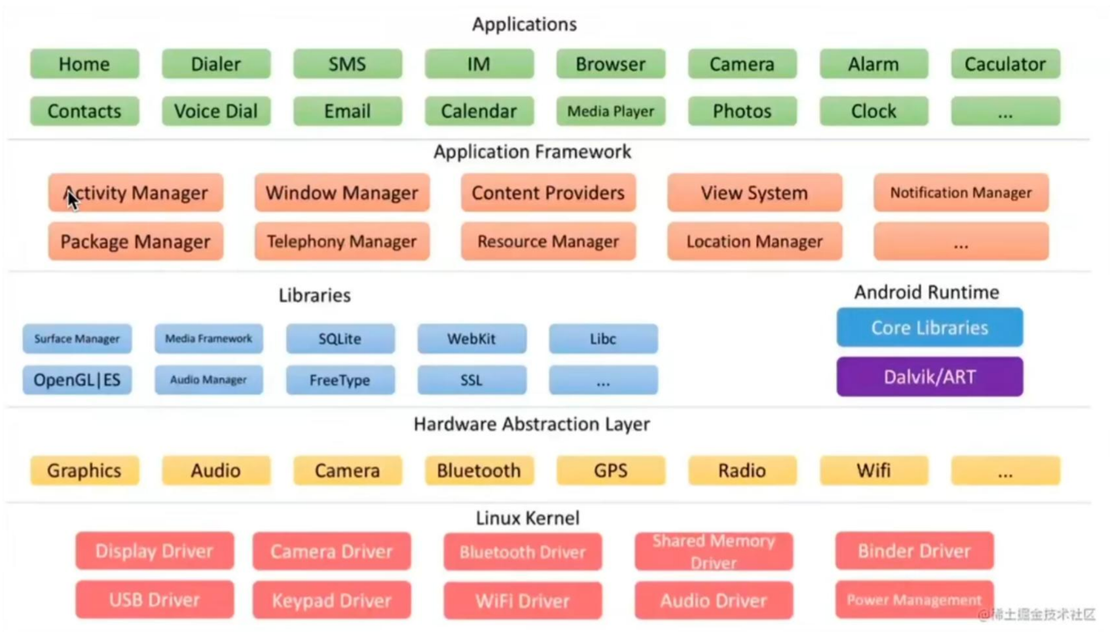
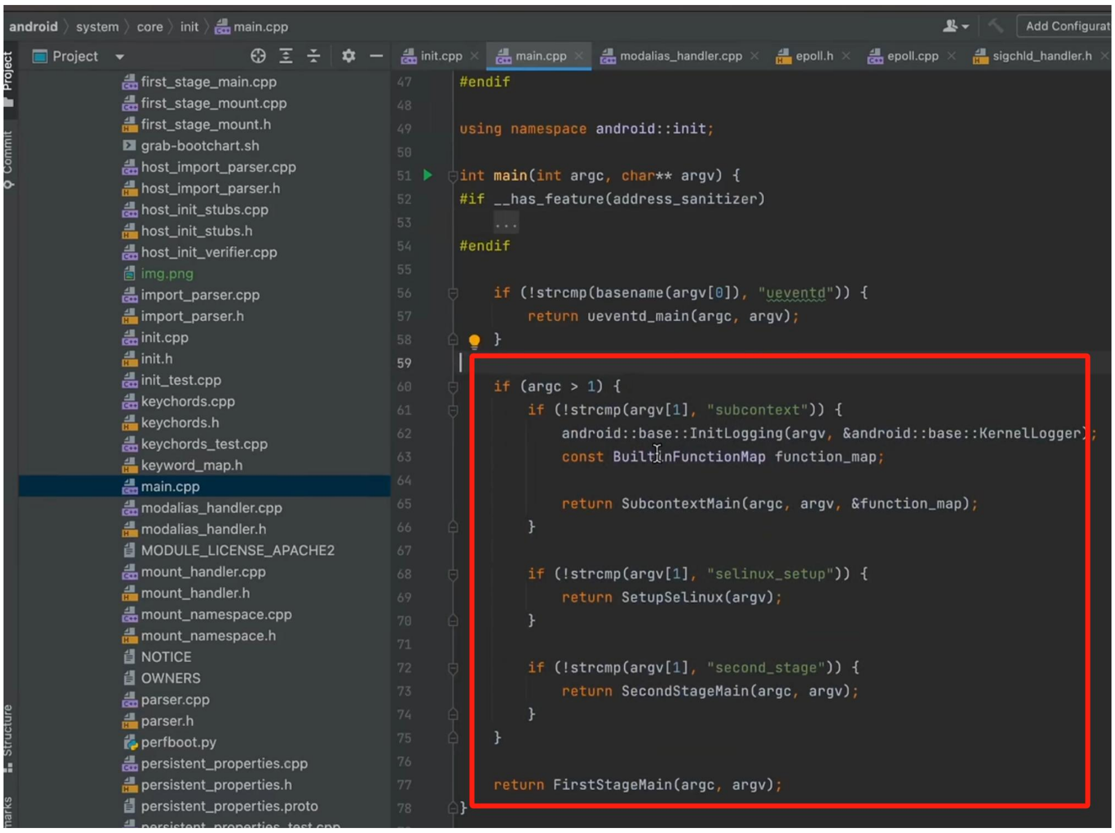
基于 Android10



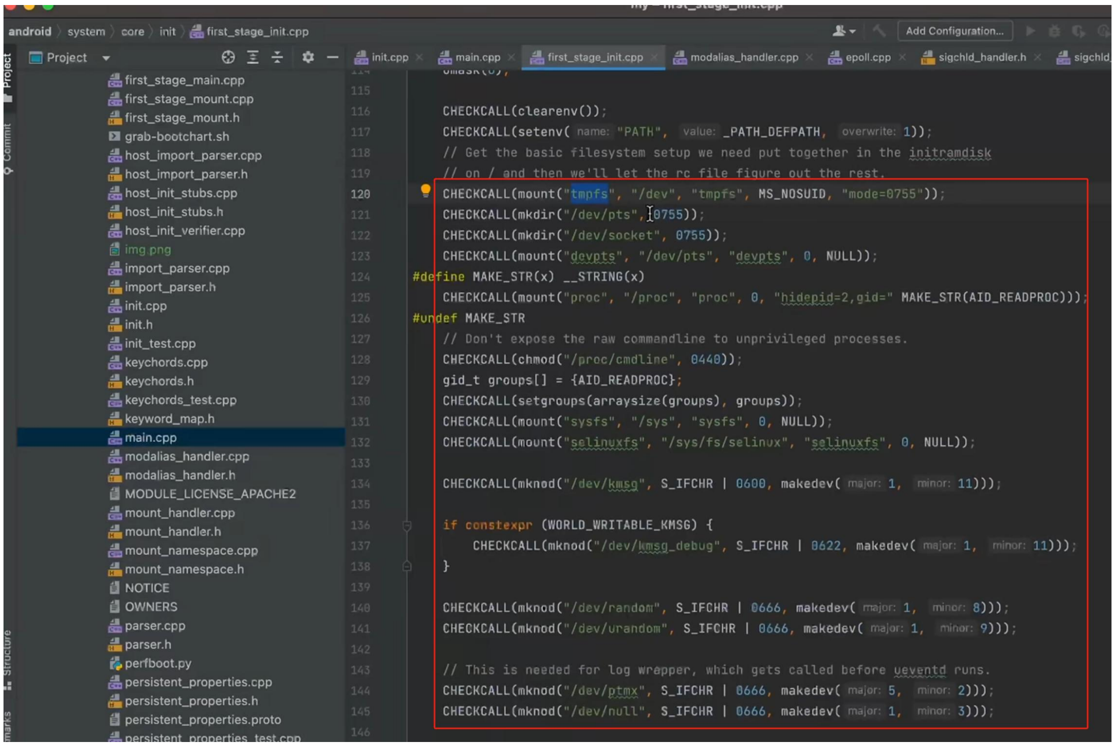
android 系统中启动的第一进程 init 进程
解析对应的 init rc 文件
入口路径如下图



first_stage_main.cpp
first_stage_mount.cpp
first_stage_mount.h
grab-bootchart.sh
host_import_parser.cpp
host_import_parser.h
host_init_stubs.cpp
host_init_stubs.h
host_init_verifier.cpp
img.png
import_parser.cpp
import_parser.h
init.cpp
init.h
init_test.cpp
keychords.cpp
keychords.h
keychords_test.cpp
keyword_map.h
main.cpp
modalias_handler.cpp
modalias_handler.h
MODULE_LICENSE_APACHE2
mount_handler.cpp
mount_handler.h
mount_namespace.cpp
mount_namespace.h
NOTICE
OWNERS
parser.cpp
parser.h
perfboot.py
persistent_properties.cpp
persistent_properties.h
persistent_properties.proto

```
#endif

using namespace android::init;

int main(int argc, char** argv) {
#if __has_feature(address_sanitizer)
    ...
#endif

    if (!strcmp(basename(argv[0]), "ueventd")) {
        return ueventd_main(argc, argv);
    }

    if (argc > 1) {
        if (!strcmp(argv[1], "subcontext")) {
            android::base::InitLogging(argv, &android::base::KernelLogger);
            const BuiltinFunctionMap function_map;

            return SubcontextMain(argc, argv, &function_map);
        }

        if (!strcmp(argv[1], "selinux_setup")) {
            return SetupSelinux(argv);
        }

        if (!strcmp(argv[1], "second_stage")) {
            return SecondStageMain(argc, argv);
        }
    }

    return FirstStageMain(argc, argv);
}
```

Firststagemain 进入



```
    umask(0);

    CHECKCALL(clearenv());
    CHECKCALL(setenv( name: "PATH",  value: _PATH_DEFPATH,  overwrite: 1));
    // Get the basic filesystem setup we need put together in the initramdisk
    // on / and then we'll let the rc file figure out the rest.
    CHECKCALL(mount("tmpfs", "/dev", "tmpfs", MS_NOSUID, "mode=0755"));
    CHECKCALL(mkdir("/dev/pts", 0755));
    CHECKCALL(mkdir("/dev/socket", 0755));
    CHECKCALL(mount("devpts", "/dev/pts", "devpts", 0, NULL));
#define MAKE_STR(x) __STRING(x)
    CHECKCALL(mount("proc", "/proc", "proc", 0, "hidepid=2,gid=" MAKE_STR(AID_READPROC)));
#undef MAKE_STR
    // Don't expose the raw commandline to unprivileged processes.
    CHECKCALL(chmod("/proc/cmdline", 0440));
    gid_t groups[] = {AID_READPROC};
    CHECKCALL(setgroups(arraysize(groups), groups));
    CHECKCALL(mount("sysfs", "/sys", "sysfs", 0, NULL));
    CHECKCALL(mount("selinuxfs", "/sys/fs/selinux", "selinuxfs", 0, NULL));

    CHECKCALL(mknod("/dev/kmsg", S_IFCHR | 0600, makedev( major: 1,  minor: 11)));

    if constexpr (WORLD_WRITABLE_KMSG) {
        CHECKCALL(mknod("/dev/kmsg_debug", S_IFCHR | 0622, makedev( major: 1,  minor: 11)));
    }

    CHECKCALL(mknod("/dev/random", S_IFCHR | 0666, makedev( major: 1,  minor: 8)));
    CHECKCALL(mknod("/dev/urandom", S_IFCHR | 0666, makedev( major: 1,  minor: 9)));

    // This is needed for log wrapper, which gets called before ueventd runs.
    CHECKCALL(mknod("/dev/ptmx", S_IFCHR | 0666, makedev( major: 5,  minor: 2)));
    CHECKCALL(mknod("/dev/null", S_IFCHR | 0666, makedev( major: 1,  minor: 3)));
```

挂载文件系统 主要的有 5 个

Tmpfs 挂载虚拟文件系统 ram 中 断电不保存

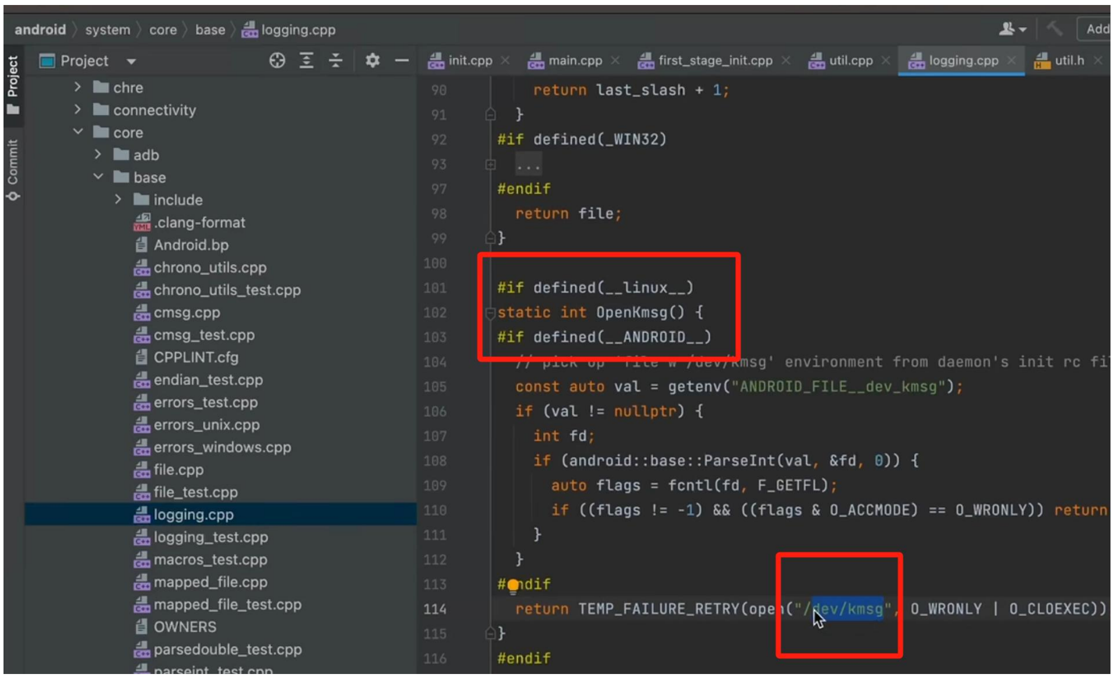
Devpts 远程虚拟终端文件设备

Proc 虚拟文件系统 可修改内核参数

Sysfs

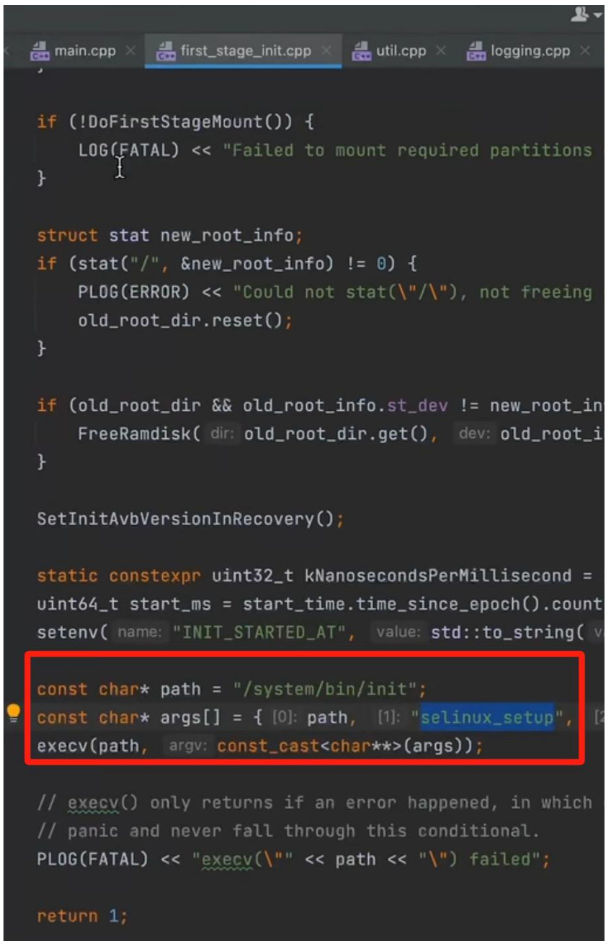
Selinuex 安全 系统每生成对象对咬经过校验检查 例如自定义的系统服务 需要 权限

挂载完文件系统 ---- 创建文件目录 --- 初始化 initlogging 日志



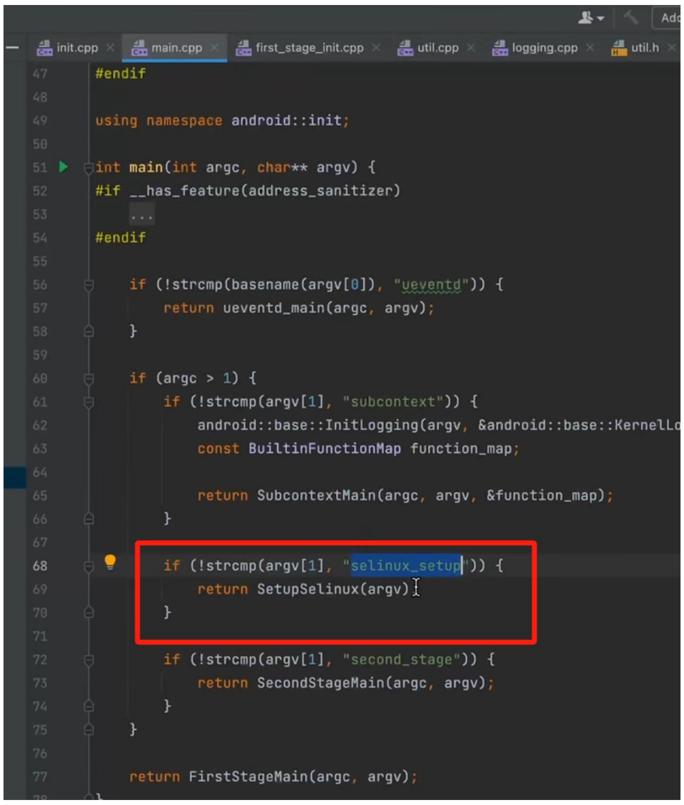
```
    return last_slash + 1;
  }
#if defined(_WIN32)
  ...
#endif
  return file;
}

#if defined(__linux__)
static int OpenKmsg() {
#if defined(__ANDROID__)
  // pick up 'file w /dev/kmsg' environment from daemon's init rc fi
  const auto val = getenv("ANDROID_FILE__dev_kmsg");
  if (val != nullptr) {
    int fd;
    if (android::base::ParseInt(val, &fd, 0)) {
      auto flags = fcntl(fd, F_GETFL);
      if ((flags != -1) && ((flags & O_ACCMODE) == O_WRONLY)) return
    }
  }
#endif
  return TEMP_FAILURE_RETRY(open("/dev/kmsg", O_WRONLY | O_CLOEXEC))
}
#endif
```

下一步调用



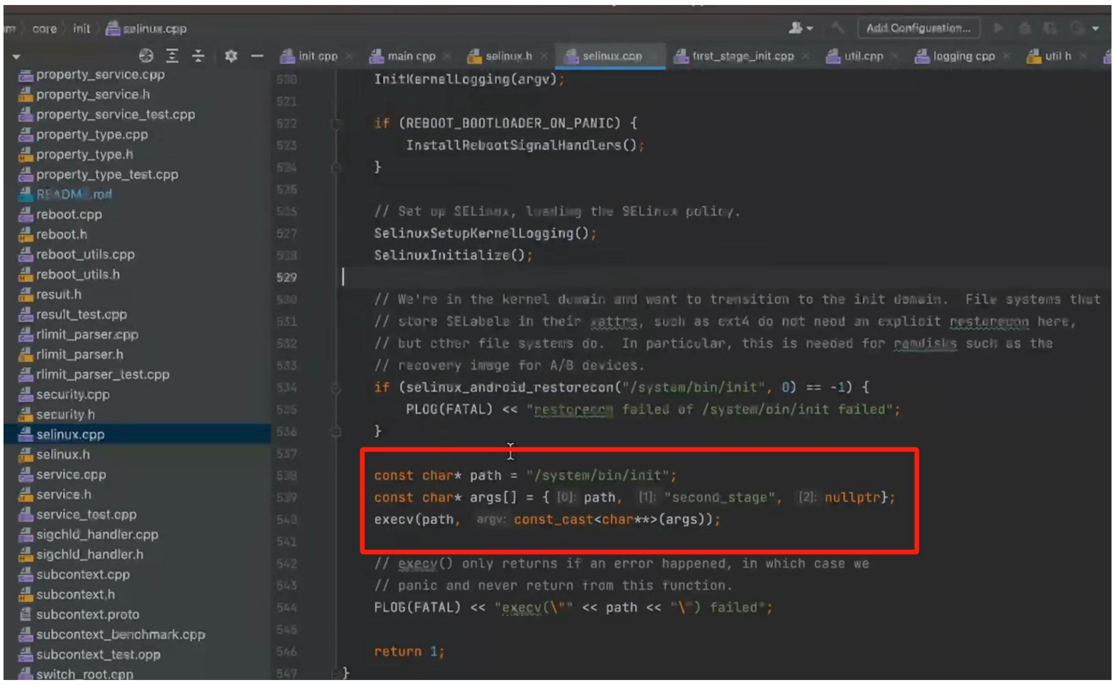
```
    if (!DoFirstStageMount()) {
        LOG(FATAL) << "Failed to mount required partitions
    }

    struct stat new_root_info;
    if (stat("/", &new_root_info) != 0) {
        PLOG(ERROR) << "Could not stat(\"/\"), not freeing
        old_root_dir.reset();
    }

    if (old_root_dir && old_root_info.st_dev != new_root_in
        FreeRamdisk( dir: old_root_dir.get(),  dev: old_root_i
    }

    SetInitAvbVersionInRecovery();

    static constexpr uint32_t kNanosecondsPerMillisecond =
    uint64_t start_ms = start_time.time_since_epoch().count
    setenv( name: "INIT_STARTED_AT",  value: std::to_string(

    const char* path = "/system/bin/init";
    const char* args[] = { [0]: path,  [1]: "selinux_setup",
    execv(path,  argv: const_cast<char**>(args));

    // execv() only returns if an error happened, in which
    // panic and never fall through this conditional.
    PLOG(FATAL) << "execv(\"" << path << "\") failed";

    return 1;
```

传递参数回到 main cpp



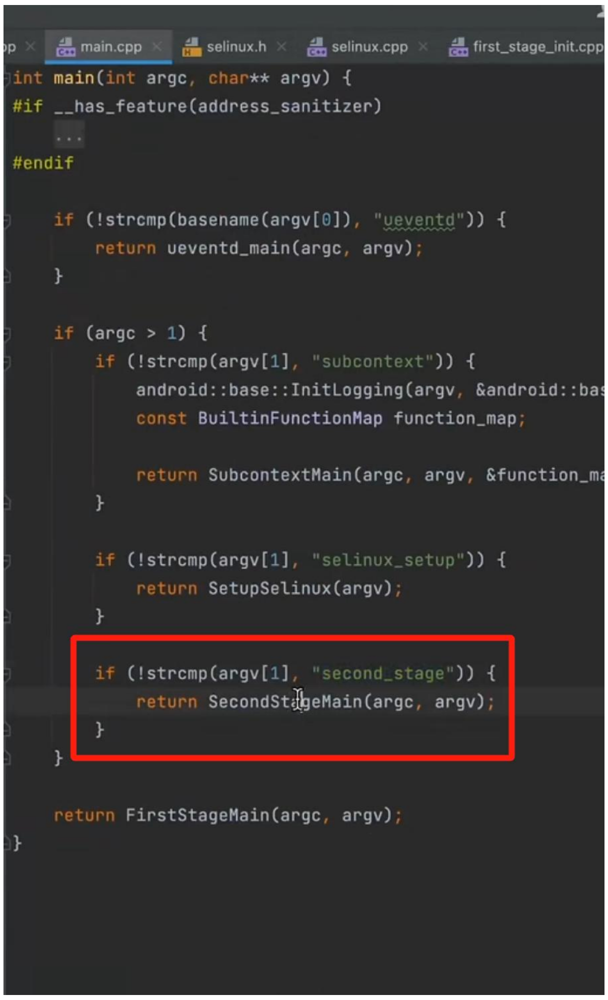
```
#endif

using namespace android::init;

int main(int argc, char** argv) {
#if __has_feature(address_sanitizer)
    ...
#endif

    if (!strcmp(basename(argv[0]), "ueventd")) {
        return ueventd_main(argc, argv);
    }

    if (argc > 1) {
        if (!strcmp(argv[1], "subcontext")) {
            android::base::InitLogging(argv, &android::base::KernelLo
            const BuiltinFunctionMap function_map;

            return SubcontextMain(argc, argv, &function_map);
        }

        if (!strcmp(argv[1], "selinux_setup")) {
            return SetupSelinux(argv);
        }

        if (!strcmp(argv[1], "second_stage")) {
            return SecondStageMain(argc, argv);
        }
    }

    return FirstStageMain(argc, argv);
```

初始化 selinuex



```
InitKernelLogging(argv);

if (REBOOT_BOOTLOADER_ON_PANIC) {
    InstallRebootSignalHandlers();
}

// Set up SELinux, loading the SELinux policy.
SelinuxSetupKernelLogging();
SelinuxInitialize();

// We're in the kernel domain and want to transition to the init domain.  File systems that
// store SELabels in their xattrs, such as ext4 do not need an explicit restorecon here,
// but other file systems do.  In particular, this is needed for ramdisks such as the
// recovery image for A/B devices.
if (selinux_android_restorecon("/system/bin/init", 0) == -1) {
    PLOG(FATAL) << "restorecon failed of /system/bin/init failed";
}

const char* path = "/system/bin/init";
const char* args[] = { path, "second_stage", nullptr};
execv(path, const_cast<char**>(args));

// execv() only returns if an error happened, in which case we
// panic and never return from this function.
PLOG(FATAL) << "execv(\"" << path << "\") failed";

return 1;
}
```

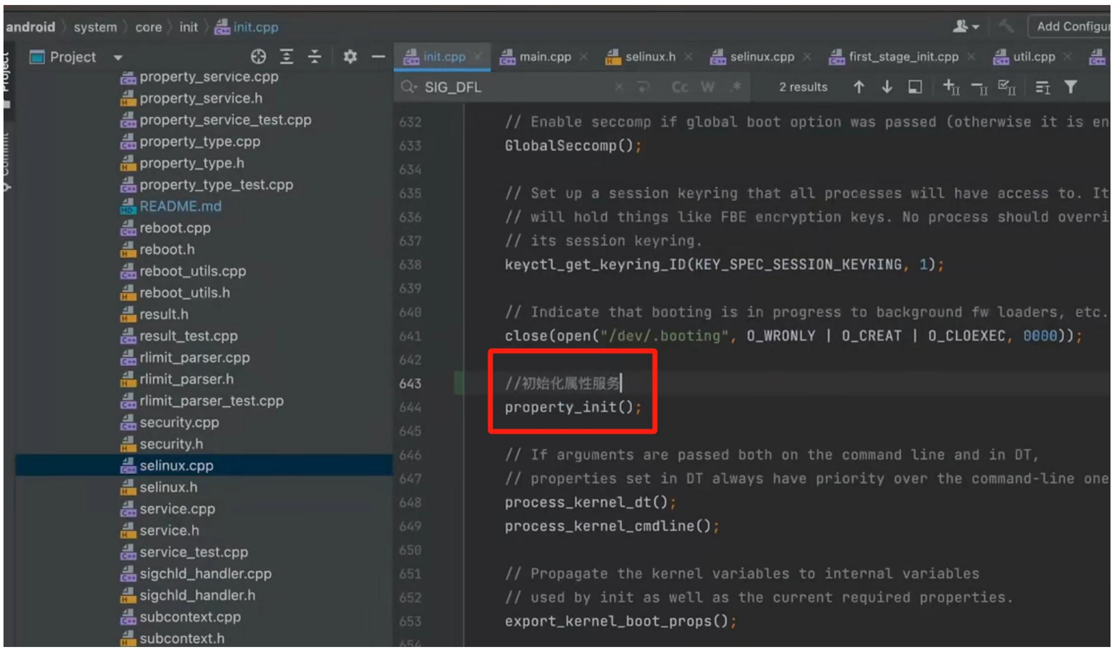
再回到 maincpp init 进程



```
int main(int argc, char** argv) {
#if __has_feature(address_sanitizer)
    ...
#endif

    if (!strcmp(basename(argv[0]), "ueventd")) {
        return ueventd_main(argc, argv);
    }

    if (argc > 1) {
        if (!strcmp(argv[1], "subcontext")) {
            android::base::InitLogging(argv, &android::bas
            const BuiltinFunctionMap function_map;

            return SubcontextMain(argc, argv, &function_ma
        }

        if (!strcmp(argv[1], "selinux_setup")) {
            return SetupSelinux(argv);
        }

        if (!strcmp(argv[1], "second_stage")) {
            return SecondStageMain(argc, argv);
        }
    }

    return FirstStageMain(argc, argv);
}
```

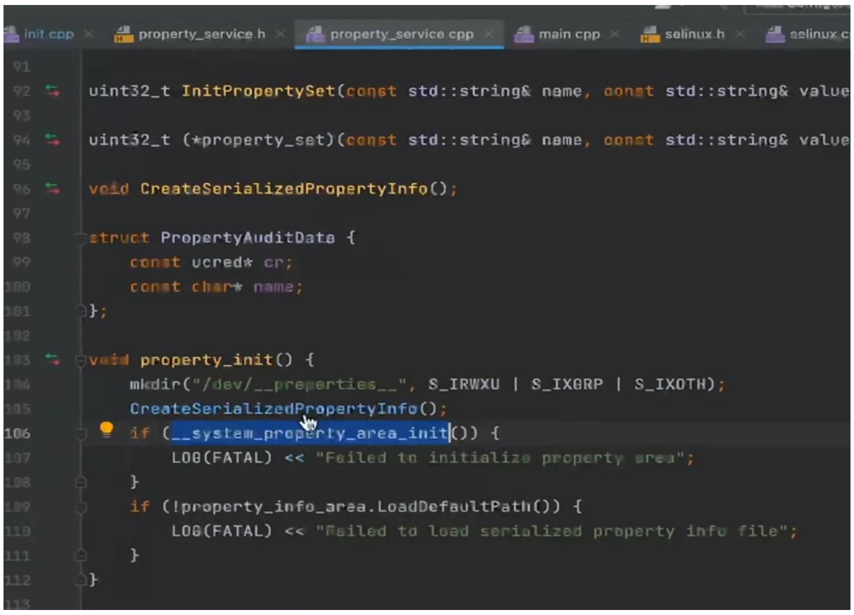
进入 second

初始化属性服务(类似与 windows 中的注册表)　系统属性配置,应用程序属性配置,手机开机读取对应的配置信息,完成初始化



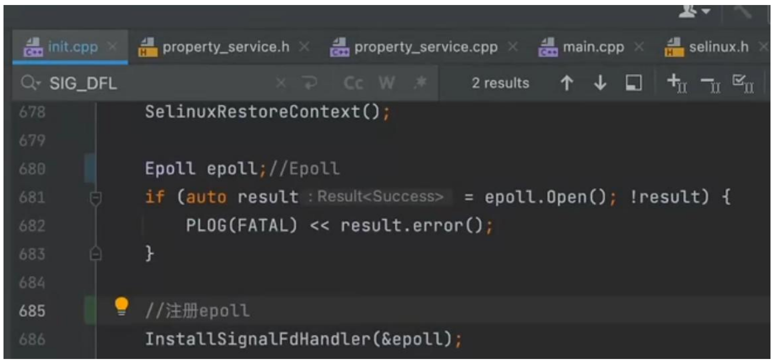
```
    // Enable seccomp if global boot option was passed (otherwise it is en
    GlobalSeccomp();

    // Set up a session keyring that all processes will have access to. It
    // will hold things like FBE encryption keys. No process should overri
    // its session keyring.
    keyctl_get_keyring_ID(KEY_SPEC_SESSION_KEYRING, 1);

    // Indicate that booting is in progress to background fw loaders, etc.
    close(open("/dev/.booting", O_WRONLY | O_CREAT | O_CLOEXEC, 0000));

    //初始化属性服务
    property_init();

    // If arguments are passed both on the command line and in DT,
    // properties set in DT always have priority over the command-line one
    process_kernel_dt();
    process_kernel_cmdline();

    // Propagate the kernel variables to internal variables
    // used by init as well as the current required properties.
    export_kernel_boot_props();
```

创建内存目录,将内存空间初始化

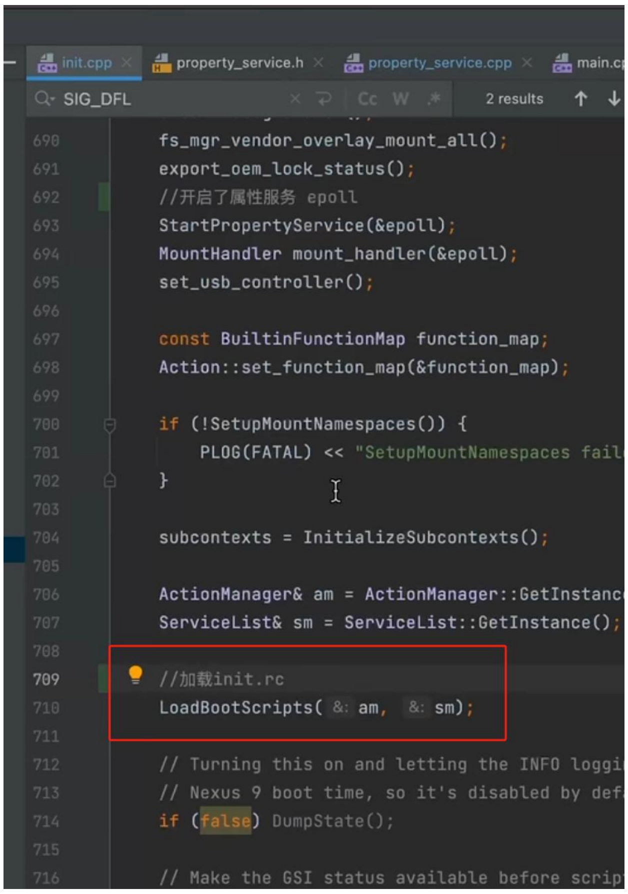


```
uint32_t InitPropertySet(const std::string& name, const std::string& value

uint32_t (*property_set)(const std::string& name, const std::string& value

void CreateSerializedPropertyInfo();

struct PropertyAuditData {
    const ucred* cr;
    const char* name;
};

void property_init() {
    mkdir("/dev/__properties__", S_IRWXU | S_IXGRP | S_IXOTH);
    CreateSerializedPropertyInfo();
    if (__system_property_area_init()) {
        LOG(FATAL) << "Failed to initialize property area";
    }
    if (!property_info_area.LoadDefaultPath()) {
        LOG(FATAL) << "Failed to load serialized property info file";
    }
}
```

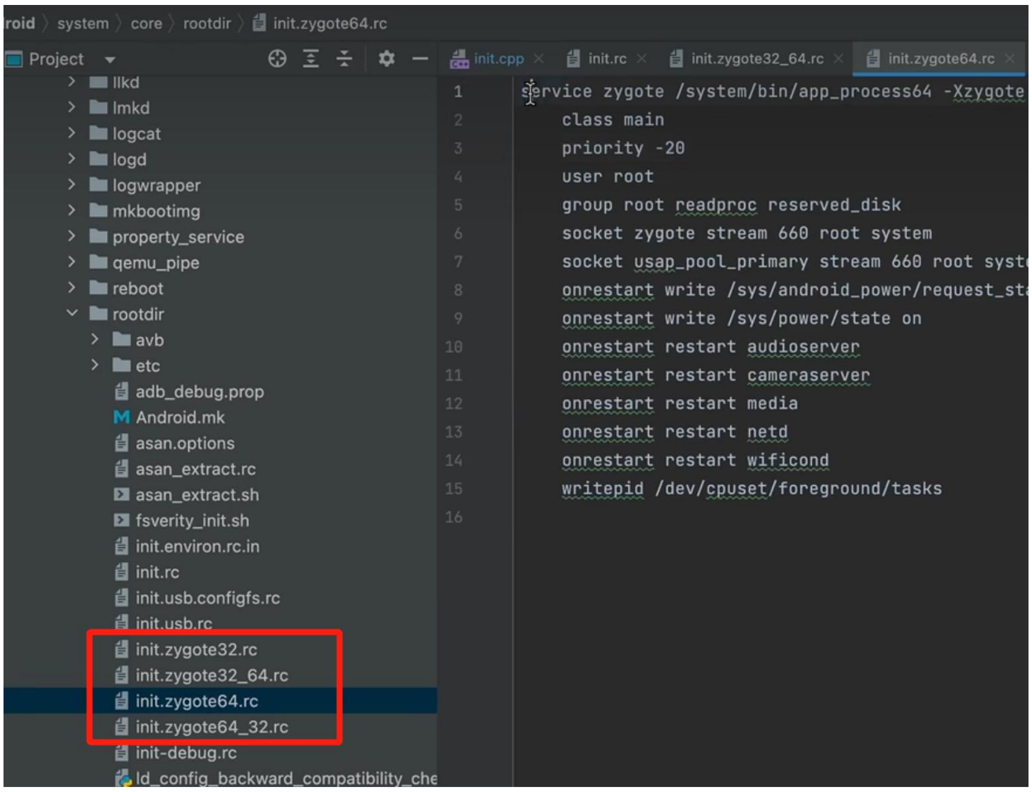
继续往下



```
    SelinuxRestoreContext();

    Epoll epoll;//Epoll
    if (auto result :Result<Success> = epoll.Open(); !result) {
        PLOG(FATAL) << result.error();
    }

    //注册epoll
    InstallSignalFdHandler(&epoll);
```



```
    fs_mgr_vendor_overlay_mount_all();
    export_oem_lock_status();
    //开启了属性服务 epoll
    StartPropertyService(&epoll);
    MountHandler mount_handler(&epoll);
    set_usb_controller();

    const BuiltinFunctionMap function_map;
    Action::set_function_map(&function_map);

    if (!SetupMountNamespaces()) {
        PLOG(FATAL) << "SetupMountNamespaces fail
    }

    subcontexts = InitializeSubcontexts();

    ActionManager& am = ActionManager::GetInstanc
    ServiceList& sm = ServiceList::GetInstance();

    //加载init.rc
    LoadBootScripts( &: am,  &: sm);

    // Turning this on and letting the INFO loggi
    // Nexus 9 boot time, so it's disabled by def
    if (false) DumpState();

    // Make the GSI status available before scrip
```

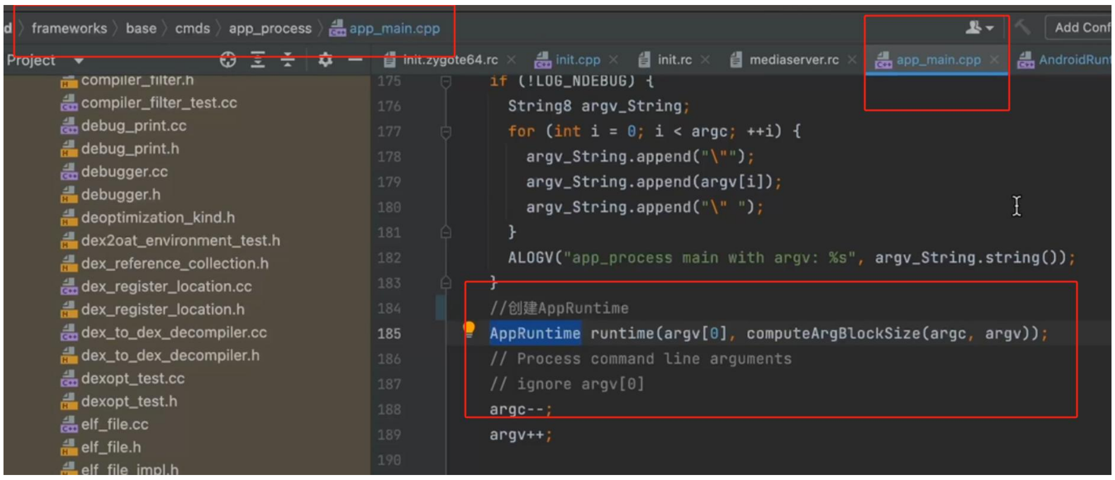


```
service zygote /system/bin/app_process64 -Xzygote
    class main
    priority -20
    user root
    group root readproc reserved_disk
    socket zygote stream 660 root system
    socket usap_pool_primary stream 660 root syste
    onrestart write /sys/android_power/request_sta
    onrestart write /sys/power/state on
    onrestart restart audioserver
    onrestart restart cameraserver
    onrestart restart media
    onrestart restart netd
    onrestart restart wificond
    writepid /dev/cpuset/foreground/tasks
```

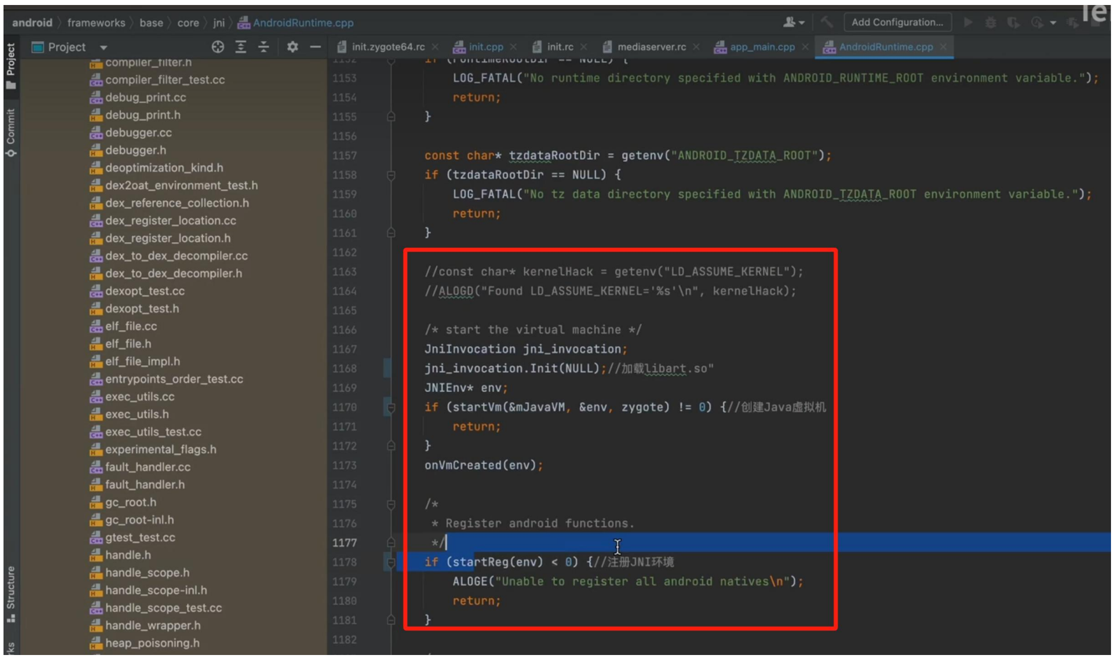
zygote 就是由 init 唤起

zygote 启动源代码入口 在 **app_main.cpp**

main 方法中 创建 appruntime -- 创建 java 虚拟机 --   注册 JNI 环境

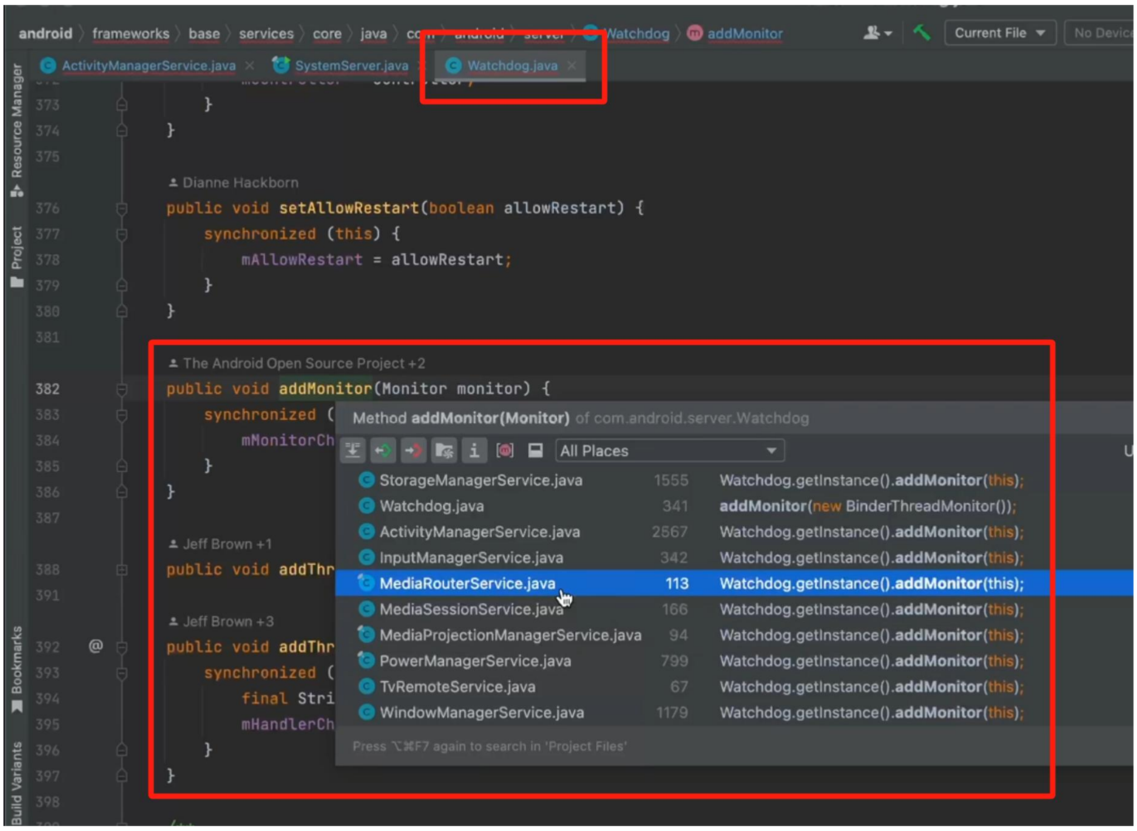


```
    if (!LOG_NDEBUG) {
      String8 argv_String;
      for (int i = 0; i < argc; ++i) {
        argv_String.append("\"");
        argv_String.append(argv[i]);
        argv_String.append("\" ");
      }
      ALOGV("app_process main with argv: %s", argv_String.string());
    }
    //创建AppRuntime
    AppRuntime runtime(argv[0], computeArgBlockSize(argc, argv));
    // Process command line arguments
    // ignore argv[0]
    argc--;
    argv++;
```



```
        LOG_FATAL("No runtime directory specified with ANDROID_RUNTIME_ROOT environment variable.");
        return;
    }

    const char* tzdataRootDir = getenv("ANDROID_TZDATA_ROOT");
    if (tzdataRootDir == NULL) {
        LOG_FATAL("No tz data directory specified with ANDROID_TZDATA_ROOT environment variable.");
        return;
    }

    //const char* kernelHack = getenv("LD_ASSUME_KERNEL");
    //ALOGD("Found LD_ASSUME_KERNEL='%s'\n", kernelHack);

    /* start the virtual machine */
    JniInvocation jni_invocation;
    jni_invocation.Init(NULL);//加载libart.so"
    JNIEnv* env;
    if (startVm(&mJavaVM, &env, zygote) != 0) {//创建Java虚拟机
        return;
    }
    onVmCreated(env);

    /*
     * Register android functions.
     */
    if (startReg(env) < 0) {//注册JNI环境
        ALOGE("Unable to register all android natives\n");
        return;
    }
```

systemserver　看门狗，监视系统服务状态



```
android 〉 frameworks 〉 base 〉 services 〉 core 〉 java 〉 com 〉 android 〉 server 〉 Watchdog 〉 addMonitor        Current File    No Device
ActivityManagerService.java ×    SystemServer.java ×    Watchdog.java ×
```

```
        }
    }

    Dianne Hackborn
    public void setAllowRestart(boolean allowRestart) {
        synchronized (this) {
            mAllowRestart = allowRestart;
        }
    }

    The Android Open Source Project +2
    public void addMonitor(Monitor monitor) {
        synchronized (
            mMonitorCh
        }
    }

    Jeff Brown +1
    public void addThr

    Jeff Brown +3
    public void addThr
        synchronized (
            final Stri
            mHandlerCh
        }
    }
```

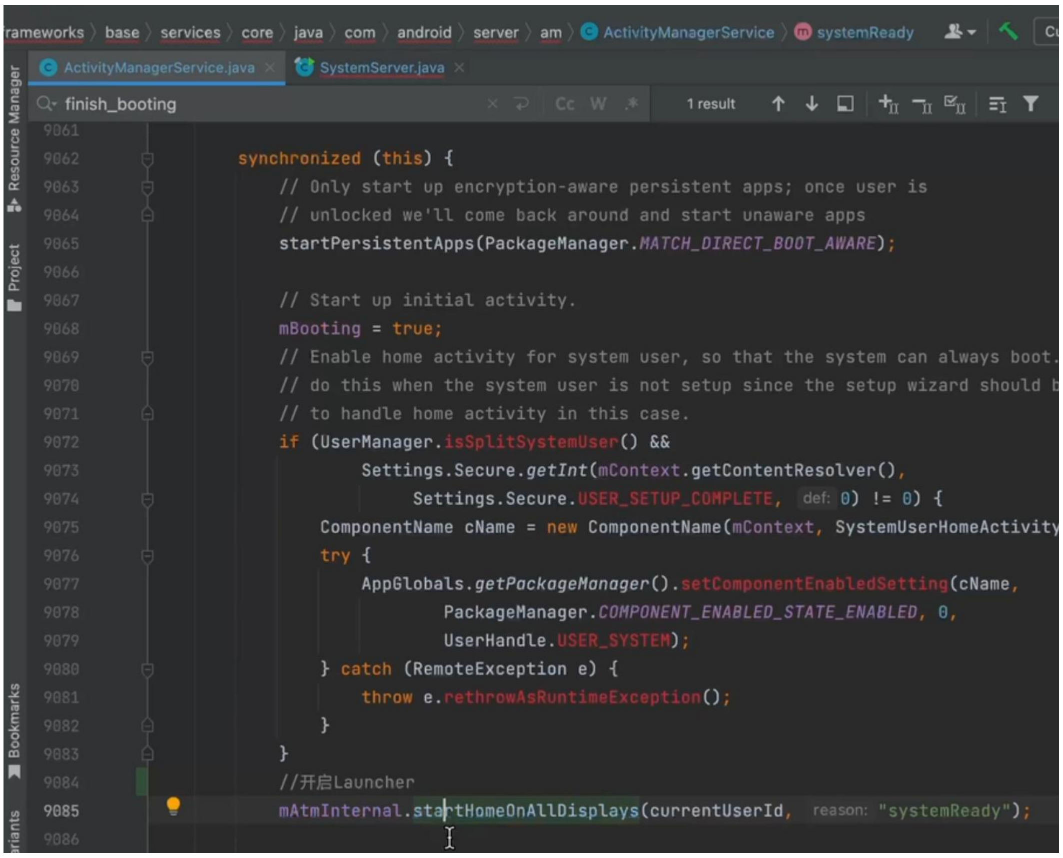
```
Method addMonitor(Monitor) of com.android.server.Watchdog
All Places
StorageManagerService.java            1555   Watchdog.getInstance().addMonitor(this);
Watchdog.java                          341   addMonitor(new BinderThreadMonitor());
ActivityManagerService.java           2567   Watchdog.getInstance().addMonitor(this);
InputManagerService.java               342   Watchdog.getInstance().addMonitor(this);
MediaRouterService.java                113   Watchdog.getInstance().addMonitor(this);
MediaSessionService.java               166   Watchdog.getInstance().addMonitor(this);
MediaProjectionManagerService.java      94   Watchdog.getInstance().addMonitor(this);
PowerManagerService.java               799   Watchdog.getInstance().addMonitor(this);
TvRemoteService.java                    67   Watchdog.getInstance().addMonitor(this);
WindowManagerService.java             1179   Watchdog.getInstance().addMonitor(this);
Press ⌥⌘F7 again to search in 'Project Files'
```

回到 **zygoteinit.java** --- startsystemserver

//服务大管家 **systemserver** -- main -- run --
开启服务
startbootstrapservices//开启引导服务
core//开启核心服务
other//开启其他服务

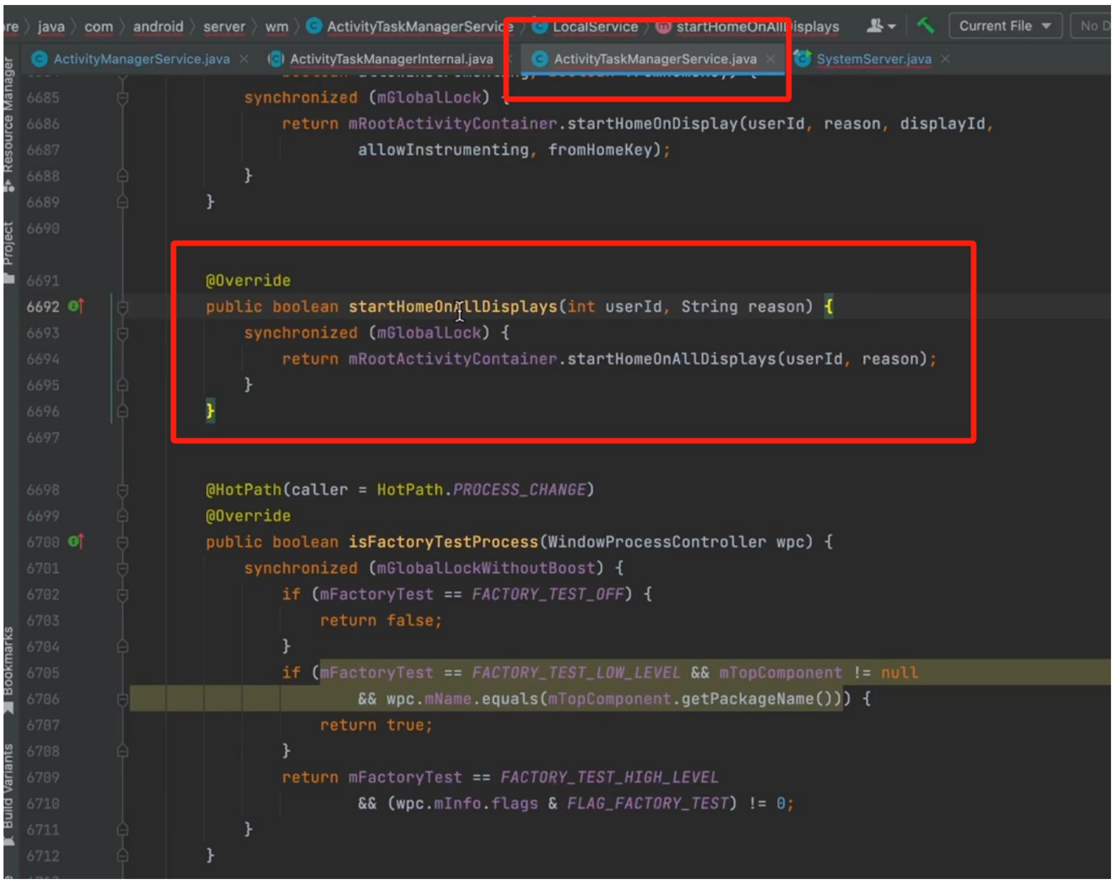
serviceManager //启动早于 zygote 源代码文件 service_manager.c
main 中 binder_open //打开 binder 驱动 映射内存空间 大小 128k

Launcher 系统启动后 第一个桌面应用　下面是开启 Launcher 调用路径

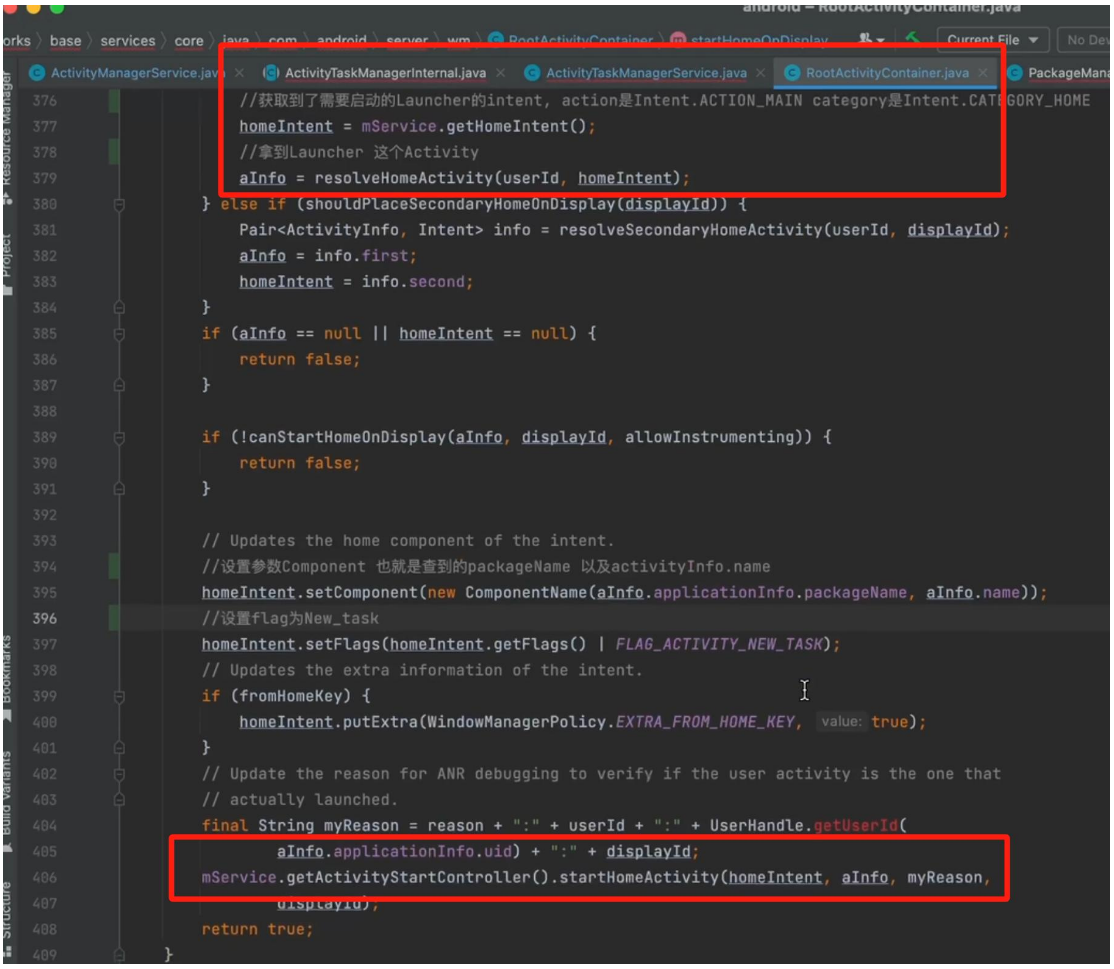

rameworks 〉 base 〉 services 〉 core 〉 java 〉 com 〉 android 〉 server 〉 am 〉 ActivityManagerService 〉 systemReady
ActivityManagerService.java
SystemServer.java
finish_booting
1 result

```
        synchronized (this) {
            // Only start up encryption-aware persistent apps; once user is
            // unlocked we'll come back around and start unaware apps
            startPersistentApps(PackageManager.MATCH_DIRECT_BOOT_AWARE);

            // Start up initial activity.
            mBooting = true;
            // Enable home activity for system user, so that the system can always boot.
            // do this when the system user is not setup since the setup wizard should b
            // to handle home activity in this case.
            if (UserManager.isSplitSystemUser() &&
                    Settings.Secure.getInt(mContext.getContentResolver(),
                        Settings.Secure.USER_SETUP_COMPLETE, def: 0) != 0) {
                ComponentName cName = new ComponentName(mContext, SystemUserHomeActivity
                try {
                    AppGlobals.getPackageManager().setComponentEnabledSetting(cName,
                            PackageManager.COMPONENT_ENABLED_STATE_ENABLED, 0,
                            UserHandle.USER_SYSTEM);
                } catch (RemoteException e) {
                    throw e.rethrowAsRuntimeException();
                }
            }
            //开启Launcher
            mAtmInternal.startHomeOnAllDisplays(currentUserId, reason: "systemReady");
```

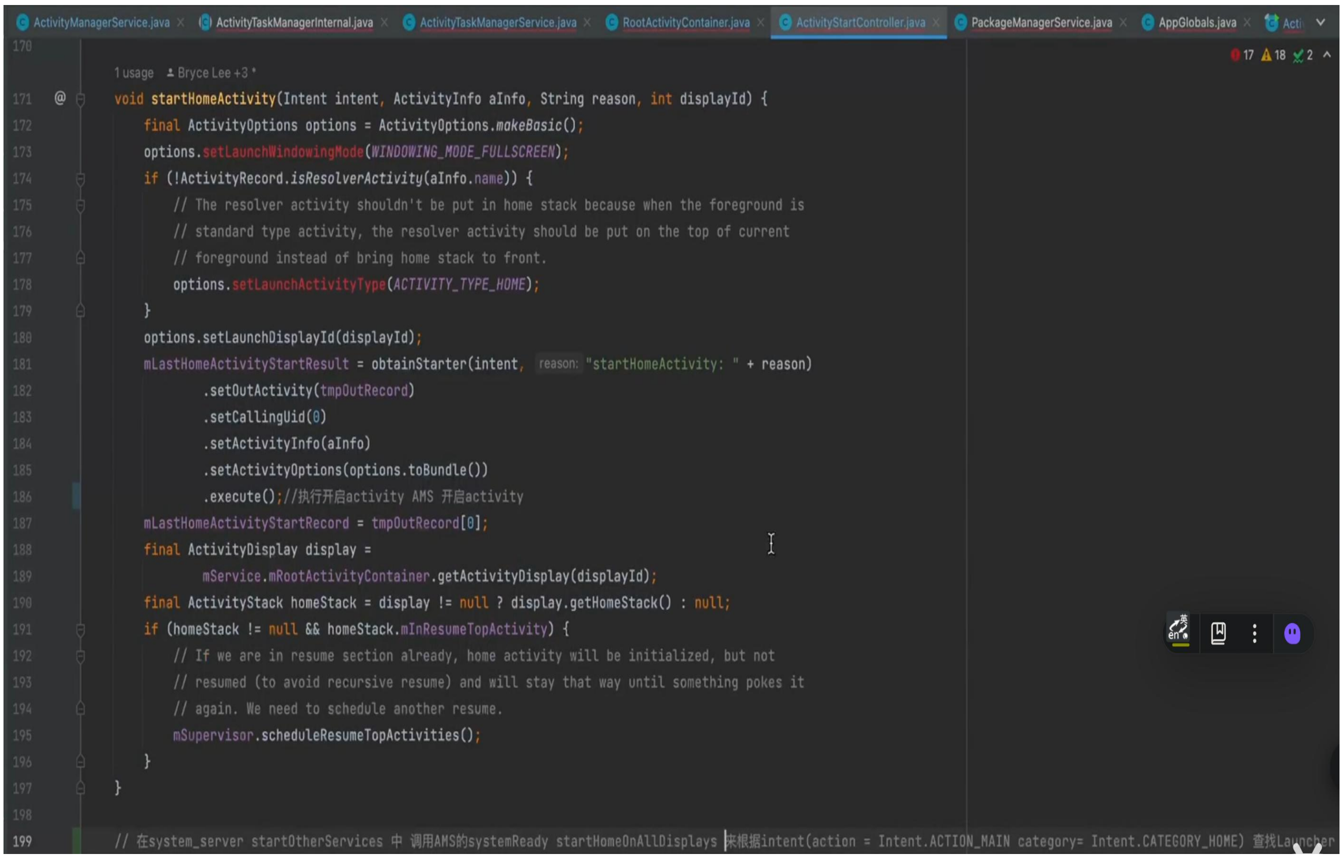
Android 10 后 ams 拆分 atms 管理 activity


re 〉 java 〉 com 〉 android 〉 server 〉 wm 〉 ActivityTaskManagerService 〉 LocalService 〉 startHomeOnAllDisplays
Current File
ActivityManagerService.java
ActivityTaskManagerInternal.java
ActivityTaskManagerService.java
SystemServer.java

```
            synchronized (mGlobalLock) {
                return mRootActivityContainer.startHomeOnDisplay(userId, reason, displayId,
                        allowInstrumenting, fromHomeKey);
            }
        }

        @Override
        public boolean startHomeOnAllDisplays(int userId, String reason) {
            synchronized (mGlobalLock) {
                return mRootActivityContainer.startHomeOnAllDisplays(userId, reason);
            }
        }

        @HotPath(caller = HotPath.PROCESS_CHANGE)
        @Override
        public boolean isFactoryTestProcess(WindowProcessController wpc) {
            synchronized (mGlobalLockWithoutBoost) {
                if (mFactoryTest == FACTORY_TEST_OFF) {
                    return false;
                }
                if (mFactoryTest == FACTORY_TEST_LOW_LEVEL && mTopComponent != null
                        && wpc.mName.equals(mTopComponent.getPackageName())) {
                    return true;
                }
                return mFactoryTest == FACTORY_TEST_HIGH_LEVEL
                        && (wpc.mInfo.flags & FLAG_FACTORY_TEST) != 0;
            }
        }
```


android – RootActivityContainer.java
orks 〉 base 〉 services 〉 core 〉 java 〉 com 〉 android 〉 server 〉 wm 〉 RootActivityContainer 〉 startHomeOnDisplay
Current File
ActivityManagerService.java
ActivityTaskManagerInternal.java
ActivityTaskManagerService.java
RootActivityContainer.java
PackageMana

```java
            //获取到了需要启动的Launcher的intent, action是Intent.ACTION_MAIN category是Intent.CATEGORY_HOME
            homeIntent = mService.getHomeIntent();
            //拿到Launcher 这个Activity
            aInfo = resolveHomeActivity(userId, homeIntent);
        } else if (shouldPlaceSecondaryHomeOnDisplay(displayId)) {
            Pair<ActivityInfo, Intent> info = resolveSecondaryHomeActivity(userId, displayId);
            aInfo = info.first;
            homeIntent = info.second;
        }
        if (aInfo == null || homeIntent == null) {
            return false;
        }

        if (!canStartHomeOnDisplay(aInfo, displayId, allowInstrumenting)) {
            return false;
        }

        // Updates the home component of the intent.
        //设置参数Component 也就是查到的packageName 以及activityInfo.name
        homeIntent.setComponent(new ComponentName(aInfo.applicationInfo.packageName, aInfo.name));
        //设置flag为New_task
        homeIntent.setFlags(homeIntent.getFlags() | FLAG_ACTIVITY_NEW_TASK);
        // Updates the extra information of the intent.
        if (fromHomeKey) {
            homeIntent.putExtra(WindowManagerPolicy.EXTRA_FROM_HOME_KEY, value: true);
        }
        // Update the reason for ANR debugging to verify if the user activity is the one that
        // actually launched.
        final String myReason = reason + ":" + userId + ":" + UserHandle.getUserId(
                aInfo.applicationInfo.uid) + ":" + displayId;
        mService.getActivityStartController().startHomeActivity(homeIntent, aInfo, myReason,
                displayId);
        return true;
    }
```


ActivityManagerService.java
ActivityTaskManagerInternal.java
ActivityTaskManagerService.java
RootActivityContainer.java
ActivityStartController.java
PackageManagerService.java
AppGlobals.java
Acti
17 18 2

```java
    1 usage  Bryce Lee +3 *
    void startHomeActivity(Intent intent, ActivityInfo aInfo, String reason, int displayId) {
        final ActivityOptions options = ActivityOptions.makeBasic();
        options.setLaunchWindowingMode(WINDOWING_MODE_FULLSCREEN);
        if (!ActivityRecord.isResolverActivity(aInfo.name)) {
            // The resolver activity shouldn't be put in home stack because when the foreground is
            // standard type activity, the resolver activity should be put on the top of current
            // foreground instead of bring home stack to front.
            options.setLaunchActivityType(ACTIVITY_TYPE_HOME);
        }
        options.setLaunchDisplayId(displayId);
        mLastHomeActivityStartResult = obtainStarter(intent, reason: "startHomeActivity: " + reason)
                .setOutActivity(tmpOutRecord)
                .setCallingUid(0)
                .setActivityInfo(aInfo)
                .setActivityOptions(options.toBundle())
                .execute();//执行开启activity AMS 开启activity
        mLastHomeActivityStartRecord = tmpOutRecord[0];
        final ActivityDisplay display =
                mService.mRootActivityContainer.getActivityDisplay(displayId);
        final ActivityStack homeStack = display != null ? display.getHomeStack() : null;
        if (homeStack != null && homeStack.mInResumeTopActivity) {
            // If we are in resume section already, home activity will be initialized, but not
            // resumed (to avoid recursive resume) and will stay that way until something pokes it
            // again. We need to schedule another resume.
            mSupervisor.scheduleResumeTopActivities();
        }
    }

    // 在system_server startOtherServices 中 调用AMS的systemReady startHomeOnAllDisplays 来根据intent(action = Intent.ACTION_MAIN category= Intent.CATEGORY_HOME) 查找Launcher
```

总结

1. Launcher 由 system_server 启动的,在 StertOtherService 中 调用了 AMs 的 systemReady ActivityTasklananarLntenel.startHame0nAllDisplays intnet action = Inent.ACTION MAIN CATEGORY = intent.CATEGORY .HOME 查找出来 launcher

2.Launcher onCreate 函数中创建了 LauncherModel startLoader 函数创建 LoadTask 去通过 Binder 访问到 LauncherAppsService 的 queryIntentActivitis 查询所有的应用信息

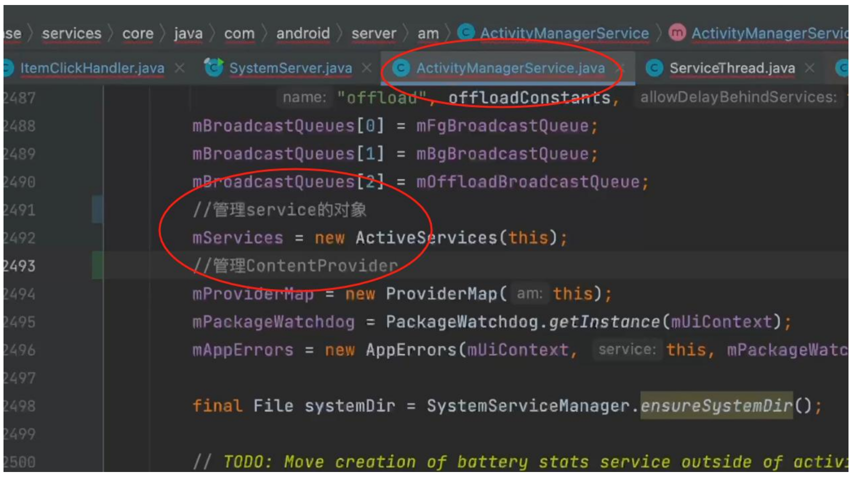
3.LoadTask 回调 OnUpdoteListener 到 rebindAdonter 对数据进行填充 绑定

4,viewHolder 中创建设置点击事件 ItemclickHandler 设量 tag 为 AppInfo 通过 tag 调用用 startAppShortcut0rInfoActivity,最终会调用到 activity 的 startActivity 函数 ams 开启 activity 的流程

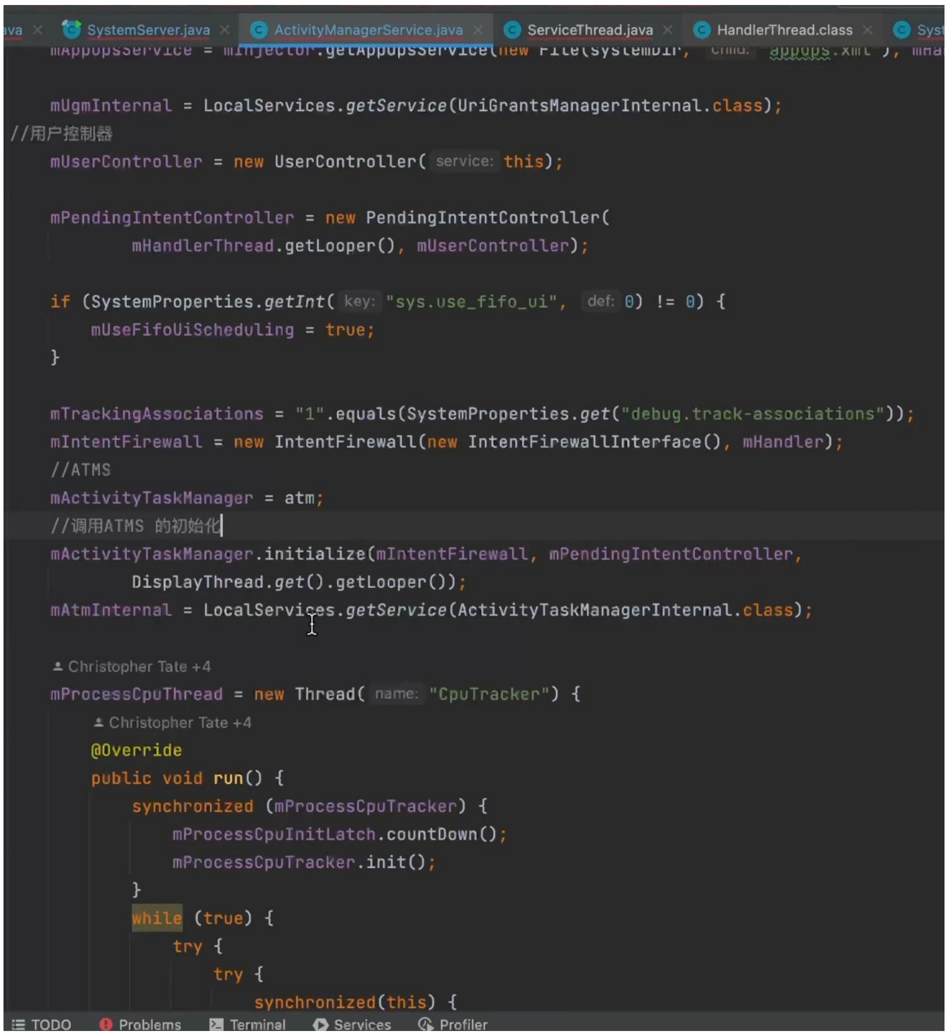
开机动画执行

AMS

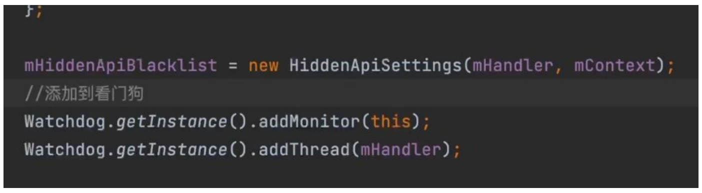
开启应用



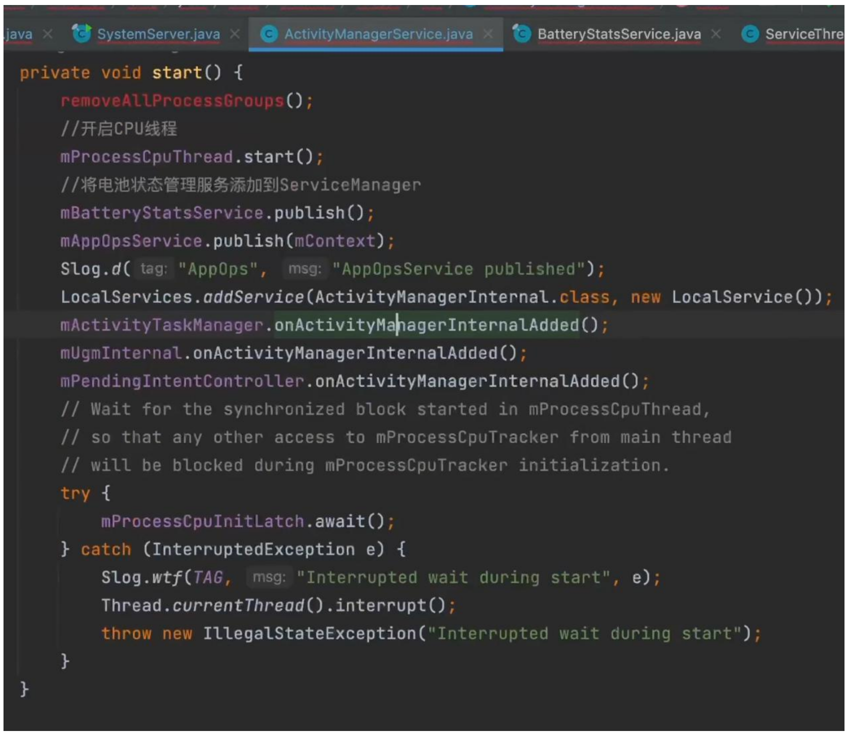
```
se 〉 services 〉 core 〉 java 〉 com 〉 android 〉 server 〉 am 〉 ActivityManagerService 〉 ActivityManagerServic
ItemClickHandler.java × SystemServer.java × ActivityManagerService.java × ServiceThread.java ×
                name: "offload", offloadConstants, allowDelayBehindServices:
        mBroadcastQueues[0] = mFgBroadcastQueue;
        mBroadcastQueues[1] = mBgBroadcastQueue;
        mBroadcastQueues[2] = mOffloadBroadcastQueue;
        //管理service的对象
        mServices = new ActiveServices(this);
        //管理ContentProvider
        mProviderMap = new ProviderMap( am: this);
        mPackageWatchdog = PackageWatchdog.getInstance(mUiContext);
        mAppErrors = new AppErrors(mUiContext,  service: this, mPackageWatc

        final File systemDir = SystemServiceManager.ensureSystemDir();

        // TODO: Move creation of battery stats service outside of activi
```



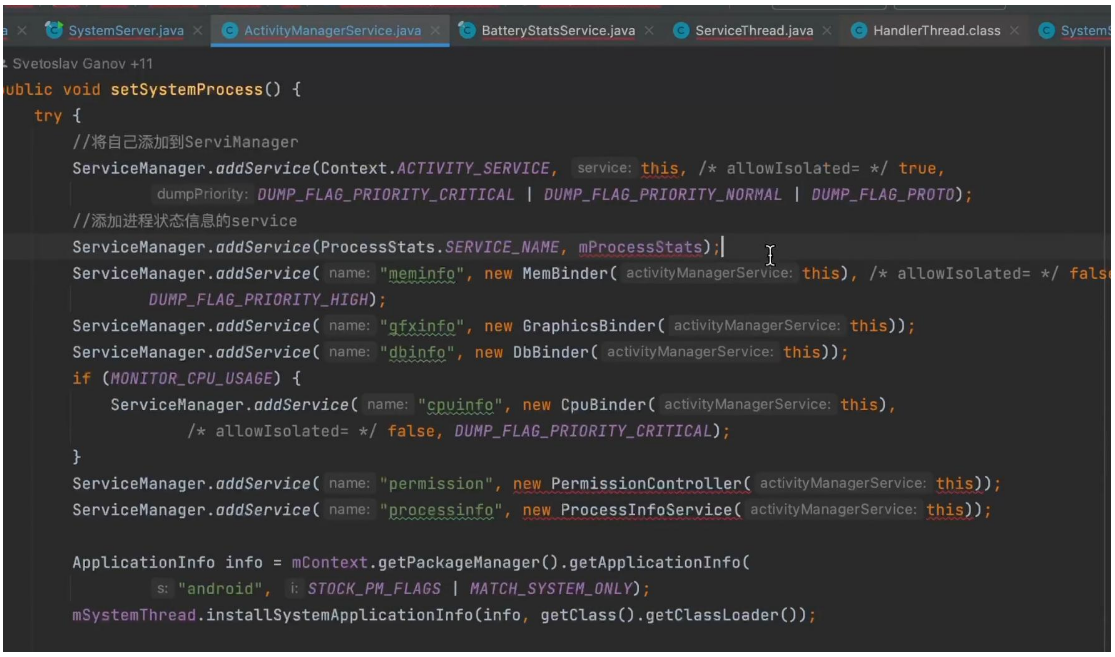
```java
    mAppOpsService = mInjector.getAppOpsService(new File(systemDir,  child: "appops.xml"), mHa

    mUgmInternal = LocalServices.getService(UriGrantsManagerInternal.class);
//用户控制器
    mUserController = new UserController( service: this);

    mPendingIntentController = new PendingIntentController(
            mHandlerThread.getLooper(), mUserController);

    if (SystemProperties.getInt( key: "sys.use_fifo_ui",  def: 0) != 0) {
        mUseFifoUiScheduling = true;
    }

    mTrackingAssociations = "1".equals(SystemProperties.get("debug.track-associations"));
    mIntentFirewall = new IntentFirewall(new IntentFirewallInterface(), mHandler);
    //ATMS
    mActivityTaskManager = atm;
    //调用ATMS 的初始化
    mActivityTaskManager.initialize(mIntentFirewall, mPendingIntentController,
            DisplayThread.get().getLooper());
    mAtmInternal = LocalServices.getService(ActivityTaskManagerInternal.class);

    Christopher Tate +4
    mProcessCpuThread = new Thread( name: "CpuTracker") {
        Christopher Tate +4
        @Override
        public void run() {
            synchronized (mProcessCpuTracker) {
                mProcessCpuInitLatch.countDown();
                mProcessCpuTracker.init();
            }
            while (true) {
                try {
                    try {
                        synchronized(this) {
```



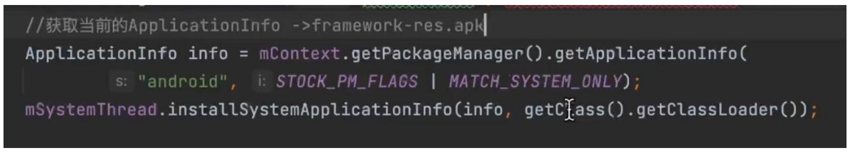
```java
  };

  mHiddenApiBlacklist = new HiddenApiSettings(mHandler, mContext);
  //添加到看门狗
  Watchdog.getInstance().addMonitor(this);
  Watchdog.getInstance().addThread(mHandler);
```



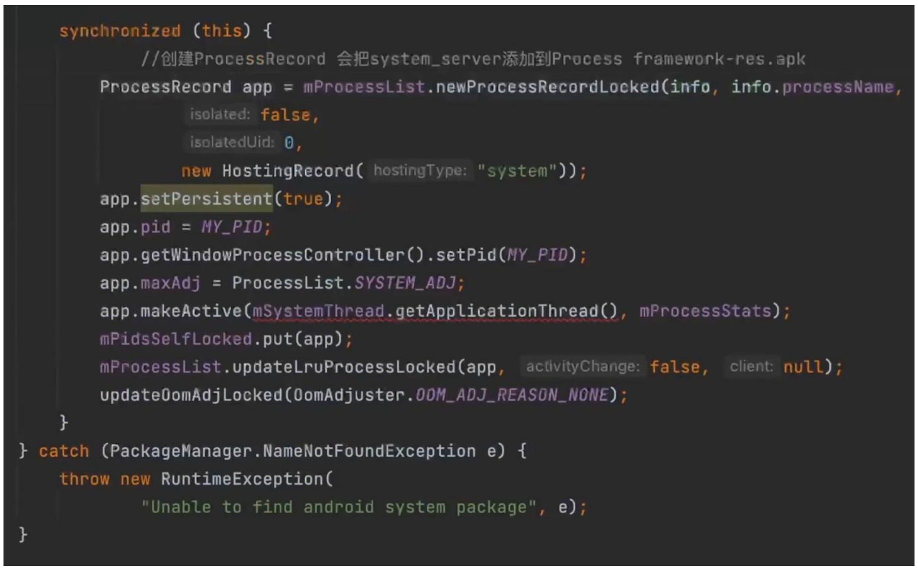
```
private void start() {
    removeAllProcessGroups();
    //开启CPU线程
    mProcessCpuThread.start();
    //将电池状态管理服务添加到ServiceManager
    mBatteryStatsService.publish();
    mAppOpsService.publish(mContext);
    Slog.d( tag: "AppOps",  msg: "AppOpsService published");
    LocalServices.addService(ActivityManagerInternal.class, new LocalService());
    mActivityTaskManager.onActivityManagerInternalAdded();
    mUgmInternal.onActivityManagerInternalAdded();
    mPendingIntentController.onActivityManagerInternalAdded();
    // Wait for the synchronized block started in mProcessCpuThread,
    // so that any other access to mProcessCpuTracker from main thread
    // will be blocked during mProcessCpuTracker initialization.
    try {
        mProcessCpuInitLatch.await();
    } catch (InterruptedException e) {
        Slog.wtf(TAG,  msg: "Interrupted wait during start", e);
        Thread.currentThread().interrupt();
        throw new IllegalStateException("Interrupted wait during start");
    }
}
```



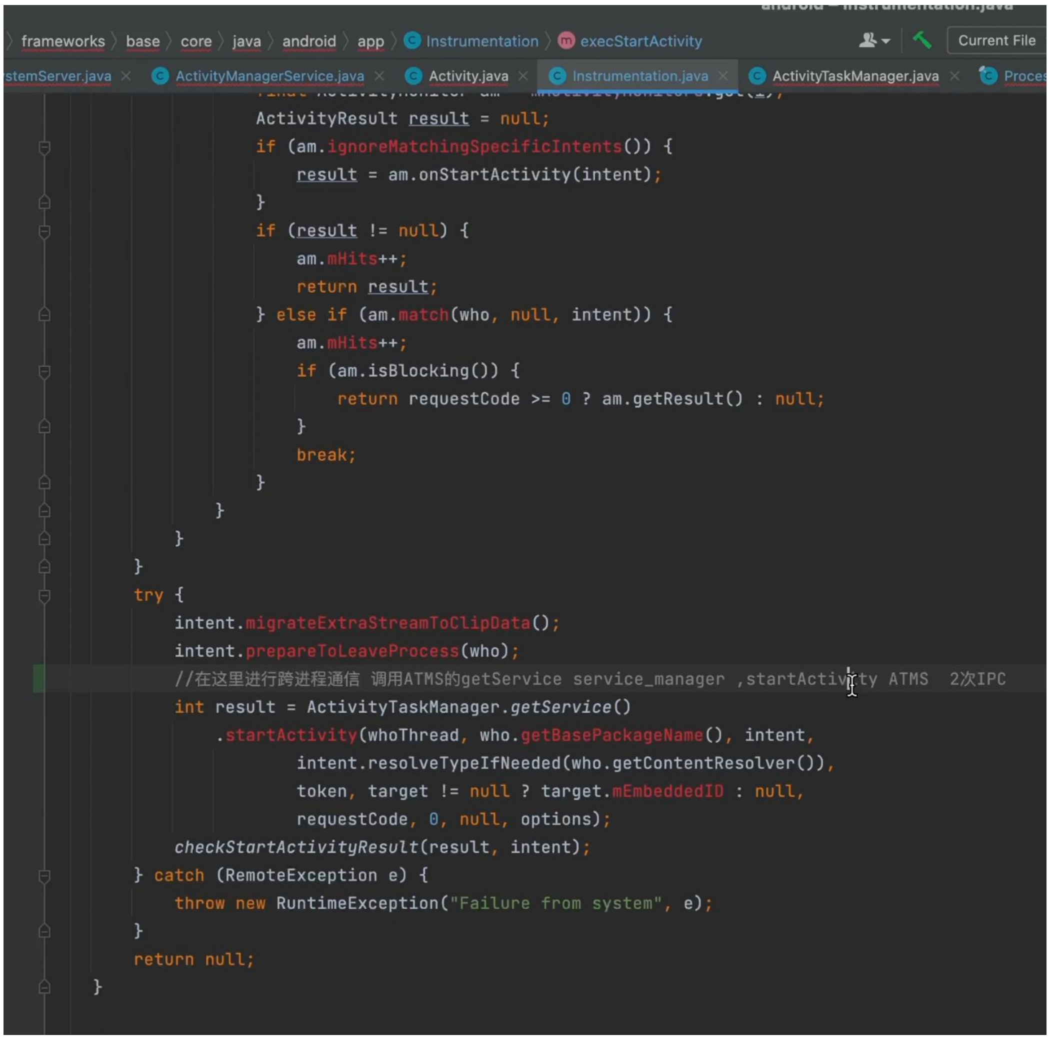
```
ublic void setSystemProcess() {
   try {
       //将自己添加到ServiManager
       ServiceManager.addService(Context.ACTIVITY_SERVICE,  service: this, /* allowIsolated= */ true,
               dumpPriority: DUMP_FLAG_PRIORITY_CRITICAL | DUMP_FLAG_PRIORITY_NORMAL | DUMP_FLAG_PROTO);
       //添加进程状态信息的service
       ServiceManager.addService(ProcessStats.SERVICE_NAME, mProcessStats);
       ServiceManager.addService( name: "meminfo", new MemBinder( activityManagerService: this), /* allowIsolated= */ fals
               DUMP_FLAG_PRIORITY_HIGH);
       ServiceManager.addService( name: "gfxinfo", new GraphicsBinder( activityManagerService: this));
       ServiceManager.addService( name: "dbinfo", new DbBinder( activityManagerService: this));
       if (MONITOR_CPU_USAGE) {
           ServiceManager.addService( name: "cpuinfo", new CpuBinder( activityManagerService: this),
                   /* allowIsolated= */ false, DUMP_FLAG_PRIORITY_CRITICAL);
       }
       ServiceManager.addService( name: "permission", new PermissionController( activityManagerService: this));
       ServiceManager.addService( name: "processinfo", new ProcessInfoService( activityManagerService: this));

       ApplicationInfo info = mContext.getPackageManager().getApplicationInfo(
                s: "android",  i: STOCK_PM_FLAGS | MATCH_SYSTEM_ONLY);
       mSystemThread.installSystemApplicationInfo(info, getClass().getClassLoader());
```



```
//获取当前的ApplicationInfo ->framework-res.apk
ApplicationInfo info = mContext.getPackageManager().getApplicationInfo(
         s: "android",  i: STOCK_PM_FLAGS | MATCH_SYSTEM_ONLY);
mSystemThread.installSystemApplicationInfo(info, getClass().getClassLoader());
```



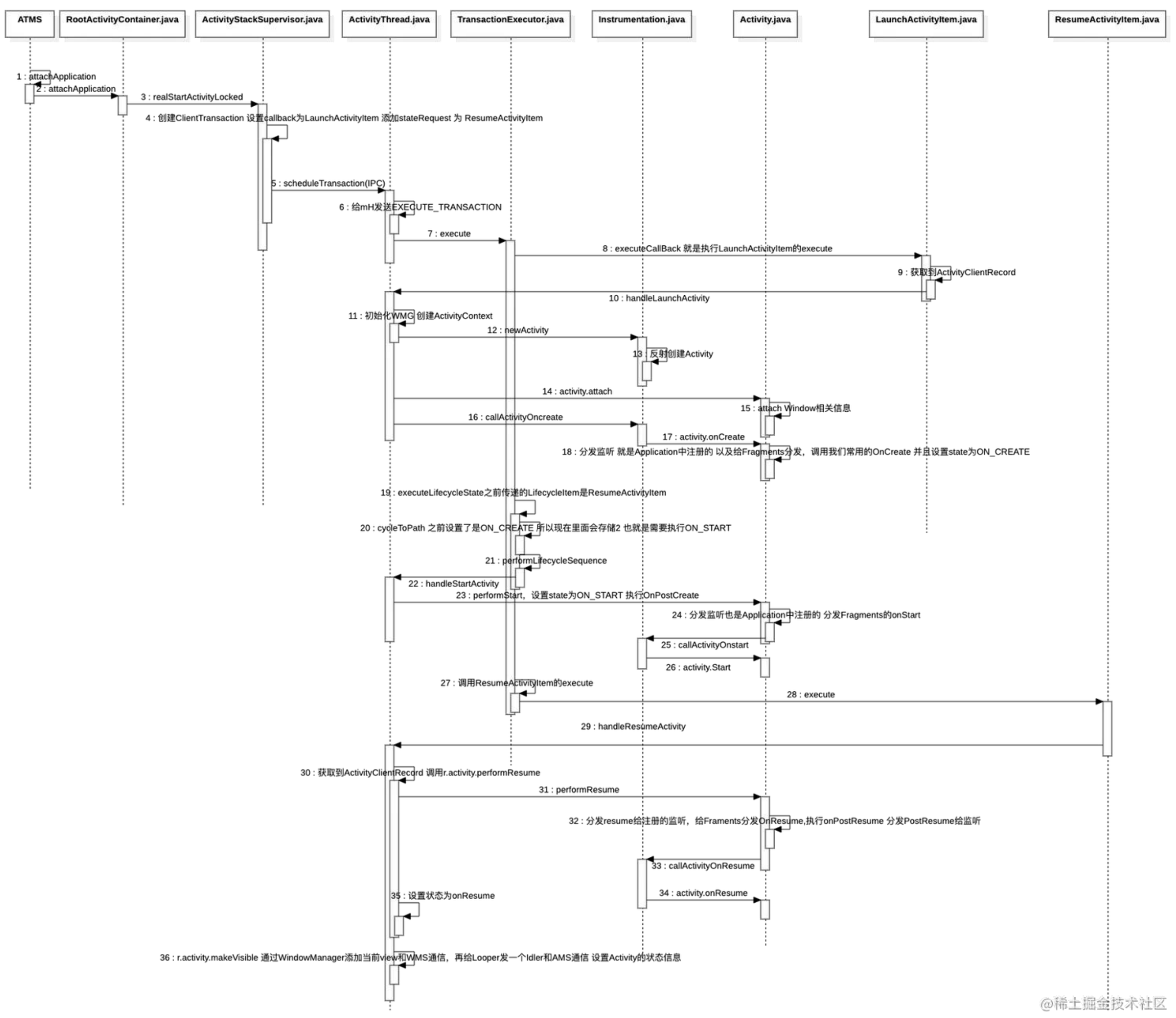
```java
    synchronized (this) {
            //创建ProcessRecord 会把system_server添加到Process framework-res.apk
        ProcessRecord app = mProcessList.newProcessRecordLocked(info, info.processName,
                isolated: false,
                isolatedUid: 0,
                new HostingRecord( hostingType: "system"));
        app.setPersistent(true);
        app.pid = MY_PID;
        app.getWindowProcessController().setPid(MY_PID);
        app.maxAdj = ProcessList.SYSTEM_ADJ;
        app.makeActive(mSystemThread.getApplicationThread(), mProcessStats);
        mPidsSelfLocked.put(app);
        mProcessList.updateLruProcessLocked(app,  activityChange: false,  client: null);
        updateOomAdjLocked(OomAdjuster.OOM_ADJ_REASON_NONE);
    }
} catch (PackageManager.NameNotFoundException e) {
    throw new RuntimeException(
            "Unable to find android system package", e);
}
```



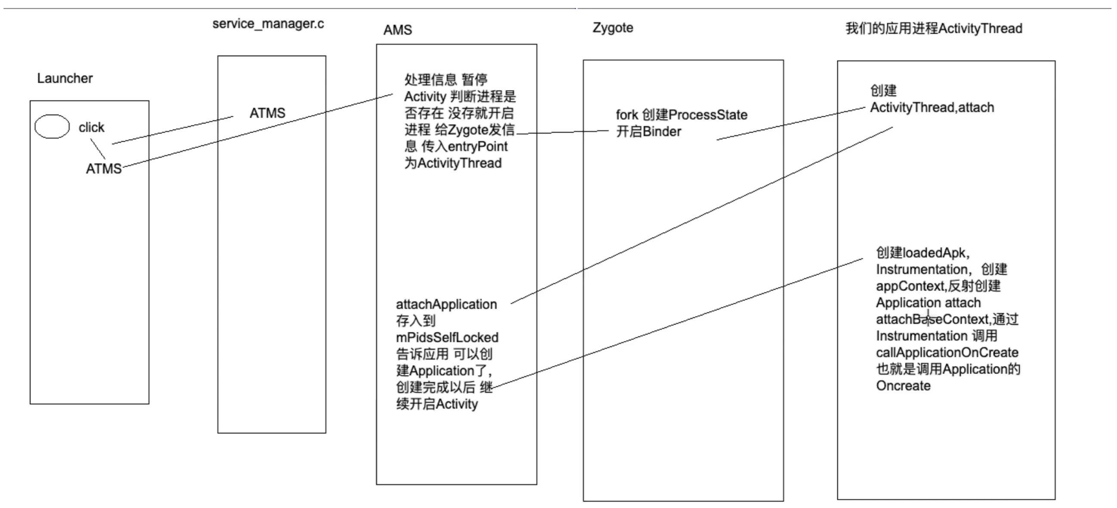
frameworks 〉 base 〉 core 〉 java 〉 android 〉 app 〉 Instrumentation 〉 execStartActivity

stemServer.java × ActivityManagerService.java × Activity.java × Instrumentation.java × ActivityTaskManager.java × Proces

```java
                    ActivityResult result = null;
                    if (am.ignoreMatchingSpecificIntents()) {
                        result = am.onStartActivity(intent);
                    }
                    if (result != null) {
                        am.mHits++;
                        return result;
                    } else if (am.match(who, null, intent)) {
                        am.mHits++;
                        if (am.isBlocking()) {
                            return requestCode >= 0 ? am.getResult() : null;
                        }
                        break;
                    }
                }
            }
        }
        try {
            intent.migrateExtraStreamToClipData();
            intent.prepareToLeaveProcess(who);
            //在这里进行跨进程通信 调用ATMS的getService service_manager ,startActivity ATMS  2次IPC
            int result = ActivityTaskManager.getService()
                .startActivity(whoThread, who.getBasePackageName(), intent,
                        intent.resolveTypeIfNeeded(who.getContentResolver()),
                        token, target != null ? target.mEmbeddedID : null,
                        requestCode, 0, null, options);
            checkStartActivityResult(result, intent);
        } catch (RemoteException e) {
            throw new RuntimeException("Failure from system", e);
        }
        return null;
    }
```



ATMS
RootActivityContainer.java
ActivityStackSupervisor.java
ActivityThread.java
TransactionExecutor.java
Instrumentation.java
Activity.java
LaunchActivityItem.java
ResumeActivityItem.java
1 : attachApplication
2 : attachApplication
3 : realStartActivityLocked
4 : 创建ClientTransaction 设置callback为LaunchActivityItem 添加stateRequest 为 ResumeActivityItem
5 : scheduleTransaction(IPC)
6 : 给mH发送EXECUTE_TRANSACTION
7 : execute
8 : executeCallBack 就是执行LaunchActivityItem的execute
9 : 获取到ActivityClientRecord
10 : handleLaunchActivity
11 : 初始化WMG 创建ActivityContext
12 : newActivity
13 : 反射创建Activity
14 : activity.attach
15 : attach Window相关信息
16 : callActivityOncreate
17 : activity.onCreate
18 : 分发监听 就是Application中注册的 以及给Fragments分发，调用我们常用的OnCreate 并且设置state为ON_CREATE
19 : executeLifecycleState之前传递的LifecycleItem是ResumeActivityItem
20 : cycleToPath 之前设置了是ON_CREATE 所以现在里面会存储2 也就是需要执行ON_START
21 : performLifecycleSequence
22 : handleStartActivity
23 : performStart，设置state为ON_START 执行OnPostCreate
24 : 分发监听也是Application中注册的 分发Fragments的onStart
25 : callActivityOnstart
26 : activity.Start
27 : 调用ResumeActivityItem的execute
28 : execute
29 : handleResumeActivity
30 : 获取到ActivityClientRecord 调用r.activity.performResume
31 : performResume
32 : 分发resume给注册的监听，给Framents分发OnResume,执行onPostResume 分发PostResume给监听
33 : callActivityOnResume
34 : activity.onResume
35 : 设置状态为onResume
36 : r.activity.makeVisible 通过WindowManager添加当前view和WMS通信，再给Looper发一个Idler和AMS通信 设置Activity的状态信息



service_manager.c
AMS
Zygote
我们的应用进程ActivityThread
Launcher
click
ATMS
ATMS
处理信息 暂停Activity 判断进程是否存在 没存在就开启进程 给Zygote发信息 传入entryPoint为ActivityThread
fork 创建ProcessState 开启Binder
创建 ActivityThread,attach
attachApplication 存入到 mPidsSelfLocked 告诉应用 可以创建Application了，创建完成以后 继续开启Activity
创建loadedApk，Instrumentation，创建appContext,反射创建Application attach attachBaseContext,通过Instrumentation 调用callApplicationOnCreate 也就是调用Application的Oncreate

Activity 启动流程