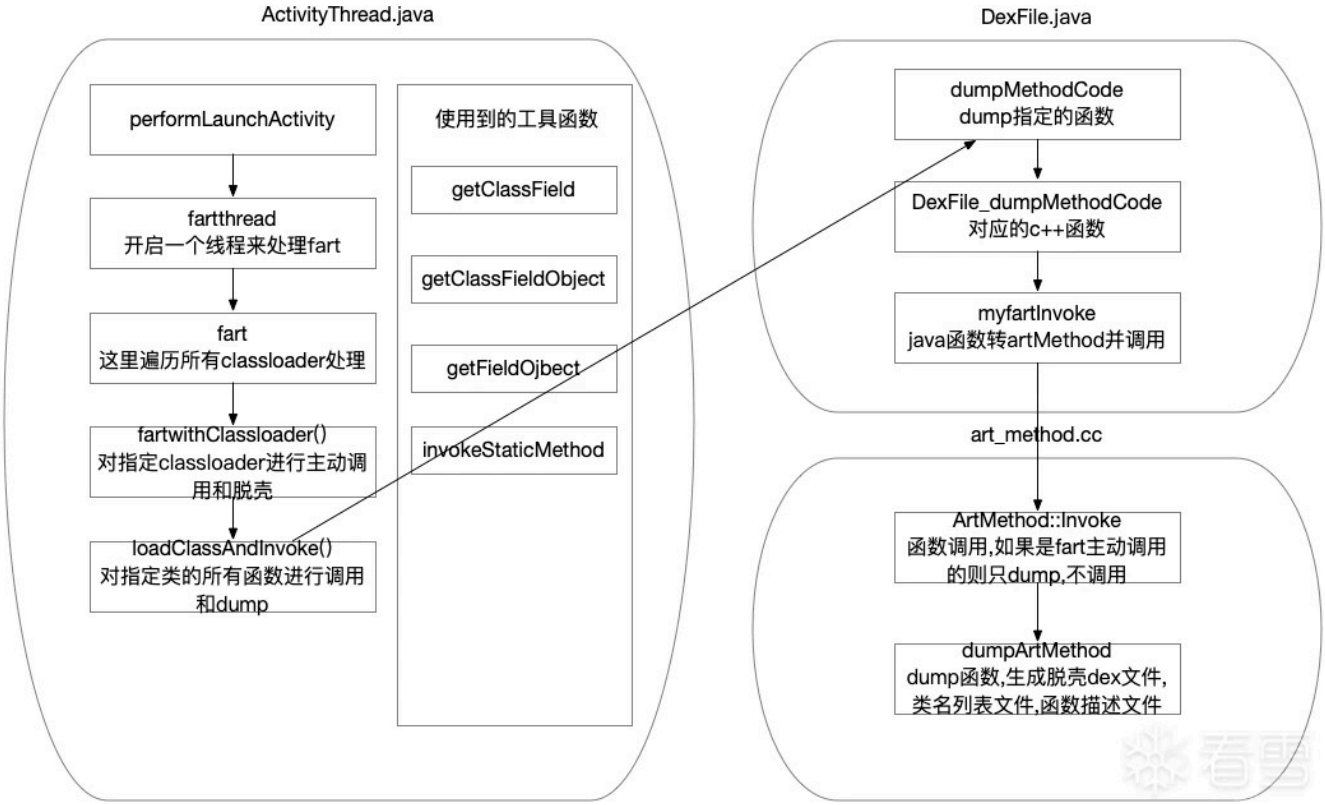

DexFile.java添加一个native函数在下面文件中实现

```
    private static native void dumpMethodCode(Object m);
```

dalvik_system_DexFile.cc添加一个函数

```
static void DexFile_dumpMethodCode(JNIEnv* env, jclass,jobject method) {
ScopedFastNativeObjectAccess soa(env);
  if(method!=nullptr)
  {
                ArtMethod* artmethod = ArtMethod::FromReflectedMethod(soa, method);
                //FromReflectedMethod 可以由Method对象得到这个方法对应的ArtMethod的真正其实地址。然后
                myfartInvoke(artmethod);
          }


  return;
}
```

ActivityThread.java是fart的入口文件

添加的工具函数

```
    public static Field getClassField(ClassLoader classloader, String class_name,String filedName)
    public static Object getClassFieldObject(ClassLoader classloader, String class_name, Object ob
    public static Object invokeStaticMethod(String class_name,String method_name, Class[] pareTypl
    public static Object getFieldOjbect(String class_name, Object obj,String filedName);
```

在performLaunchActivity函数末尾添加fart启动线程函数： fartthread();
fart执行链
performLaunchActivity()-> public static void fartthread()-> public static void fart()-> public static ClassLoader getClassloader()->
public static void loadClassAndInvoke(ClassLoader appClassloader, String eachclassname, Method dumpMethodCode_method);

```
    public static void loadClassAndInvoke(ClassLoader appClassloader, String eachclassname, Metho
    //获取dumpMethodCode_method的构造函数和普通函数列表并且调用dumpMethodCode_method.invoke()

     public static ClassLoader getClassloader();
     //通过反射获取classloader
     //android.app.ActivityThread(currentActivityThread)->(mBoundApplication)->(android.app.Activ
    public static void fart();
    //获取classloader的dexfile的getclassNamelist函数拿到dex对象的类列表然后给loadClassAndInvoke调用。
```

art_method.cc

```
        extern "C" void myfartInvoke(ArtMethod * artmethod)
         SHARED_LOCKS_REQUIRED(Locks::mutator_lock_) {
                JValue *result = nullptr;
                Thread *self = nullptr;
                uint32_t temp = 6;
                uint32_t *args = &temp;
                uint32_t args_size = 6;
                artmethod->Invoke(self, args, args_size, result, "fart");
        }

extern "C" void dumpArtMethod(ArtMethod * artmethod) SHARED_LOCKS_REQUIRED(Locks::mutator_lock_)
extern "C" void dumpDexFileByExecute(ArtMethod * artmethod) SHARED_LOCKS_REQUIRED(Locks::mutator_l
```

```
extern "C" char *base64_encode(char *str, long str_len,
                long *outlen) {
long len;
char *res;
int i, j;
const char *base64_table =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
if (str_len % 3 == 0)
    len = str_len / 3 * 4;
else
    len = (str_len / 3 + 1) * 4;

res = (char *) malloc(sizeof(char) * (len + 1));
res[len] = '\0';
*outlen = len;
for (i = 0, j = 0; i < len - 2; j += 3, i += 4) {
    res[i] = base64_table[str[j] >> 2];
    res[i + 1] =
        base64_table[(str[j] & 0x3) << 4 |
                (str[j + 1] >> 4)];
    res[i + 2] =
        base64_table[(str[j + 1] & 0xf) << 2 |
                (str[j + 2] >> 6)];
    res[i + 3] = base64_table[str[j + 2] & 0x3f];
}

switch (str_len % 3) {
case 1:
    res[i - 2] = '=';
    res[i - 1] = '=';
    break;
case 2:
    res[i - 1] = '=';
    break;
}

return res;
}
```

```
uint8_t *codeitem_end(const uint8_t ** pData) {
    uint32_t num_of_list = DecodeUnsignedLeb128(pData);
    for (; num_of_list > 0; num_of_list--) {
        int32_t num_of_handlers =
            DecodeSignedLeb128(pData);
        int num = num_of_handlers;
        if (num_of_handlers <= 0) {
            num = -num_of_handlers;
        }
        for (; num > 0; num--) {
            DecodeUnsignedLeb128(pData);
            DecodeUnsignedLeb128(pData);
        }
        if (num_of_handlers <= 0) {
            DecodeUnsignedLeb128(pData);
        }
    }
    return (uint8_t *) (*pData);
}
```

iterpreter.cc

```
static JValue Execute(Thread* self, const DexFile::CodeItem* code_item, ShadowFrame& shadow_frame,
                      JValue result_register)
    SHARED_LOCKS_REQUIRED(Locks::mutator_lock_);

static inline JValue Execute(Thread* self, const DexFile::CodeItem* code_item,
                             ShadowFrame& shadow_frame, JValue result_register) {


  if(strstr(PrettyMethod(shadow_frame.GetMethod()).c_str(),"<clinit>")!=nullptr)
  {
          dumpDexFileByExecute(shadow_frame.GetMethod());
          }
```