SOM-ITSOLUTIONS

Android

Android Service Internals

SOMENATH MUKHOPADHYAY

som-itsolutions
#A2 1/13 South Purbachal Hospital Road Kolkata 700078 Mob: +91 9748185282
Email: som@som-itsolutions.com / som.mukhopadhyay@gmail.com
Website: http://www.som-itsolutions.com/
Blog: www.som-itsolutions.blogspot.com

Have you ever wondered how an app gets an handle to the system services like POWER MANAGER or ACTIVITY MANAGER or LOCATION MANAGER and several others like these. To know that I dug into the source code of Android and found out how this is done internally.

The investigation detailed below will work as an hand-holder for an Android internal learners.

So let me start from the application side's java code.

At the application side we have to call the function getService and pass the ID of the system service (say POWER_SERVCE) to get an handle to the service.

Here is the code for getService defined in /frameworks/base/core/java/android/os/ServiceManager.java

```
/**
44     * Returns a reference to a service with the given name.
45     *
46     * @param name the name of the service to get
47     * @return a reference to the service, or <code>null</code> if the service doesn't
exist
48     */
49    public static IBinder getService(String name) {
50        try {
51            IBinder service = sCache.get(name);
52            if (service != null) {
53                return service;
54            } else {
55                return getIServiceManager().getService(name);
56            }
57        } catch (RemoteException e) {
58            Log.e(TAG, "error in getService", e);
59        }
60        return null;
61    }
```

Suppose we don't have the service in the cache. Hence we need to concentrate on the line 55
return getIServiceManager().getService(name);

This call actually gets an handle to the service manager and asks it to return a reference of the service whose name we have passed as a parameter.

Now let us see how the getIServiceManager() function returns a handle to the ServiceManager.

Here is the code of getIserviceManager() from /frameworks/base/core/java/android/os/ServiceManager.java

```
private static IServiceManager getIServiceManager() {
34        if (sServiceManager != null) {
35            return sServiceManager;
36        }
37
38        // Find the service manager
39        sServiceManager =
ServiceManagerNative.asInterface(BinderInternal.getContextObject());
40        return sServiceManager;
41    }
```

Look at the line 39. Here we get an handle to the BpServiceManager. The reason is because after the systemserver starts servicemanager(call main in service_manager.c), the servicemanager will register itself as a context_manager of binder by ioctl(bs->fd, BINDER_SET_CONTEXT_MGR, 0) through the function

```
int binder_become_context_manager(struct binder_state *bs)
{
return ioctl(bs->fd, BINDER_SET_CONTEXT_MGR, 0);
}
```

The **ServicemanagerNative.asInterface()** looks like the following:

```
/**
28     * Cast a Binder object into a service manager interface, generating
29     * a proxy if needed.
30     */
31    static public IServiceManager asInterface(IBinder obj)
32    {
33        if (obj == null) {
34            return null;
35        }
36        IServiceManager in =
37            (IServiceManager)obj.queryLocalInterface(descriptor);
38        if (in != null) {
39            return in;
40        }
41
42        return new ServiceManagerProxy(obj);
43    }
```

So basically we are getting a handle to the native servicemanager.

This asInterface function is actually buried inside the two macros DECLARE_META_INTERFACE(ServiceManager) and IMPLEMENT_META_INTERFACE(ServiceManager, "android.os.IServiceManager"); defined in IserviceManager.h and IServiceManager.cpp respectively.

Lets delve into the two macros defined in /frameworks/base/include/binder/IInterface.h

DECLARE_META_INTERFACE(ServiceManager) macro.

Its defined as

```
// ----------------------------------------------------------------------
73
74#define DECLARE_META_INTERFACE(INTERFACE)                               \
75    static const android::String16 descriptor;                        \
76    static android::sp<I##INTERFACE> asInterface(                     \
77            const android::sp<android::IBinder>& obj);                \
```

```
78    virtual const android::String16& getInterfaceDescriptor() const;    \
79    I##INTERFACE();                                                      \
80    virtual ~I##INTERFACE();                                             \
```

And the IMPLEMENT_META_INTERFACE(ServiceManager, "android.os.IServiceManager");
has been defined as follows:

```
#define IMPLEMENT_META_INTERFACE(INTERFACE, NAME)                          \
84    const android::String16 I##INTERFACE::descriptor(NAME);              \
85    const android::String16&                                             \
86            I##INTERFACE::getInterfaceDescriptor() const {               \
87        return I##INTERFACE::descriptor;                                 \
88    }                                                                    \
89    android::sp<I##INTERFACE> I##INTERFACE::asInterface(                 \
90            const android::sp<android::IBinder>& obj)                    \
91    {                                                                    \
92        android::sp<I##INTERFACE> intr;                                  \
93        if (obj != NULL) {                                               \
94            intr = static_cast<I##INTERFACE*>(                           \
95                obj->queryLocalInterface(                                \
96                        I##INTERFACE::descriptor).get());                \
97            if (intr == NULL) {                                          \
98                intr = new Bp##INTERFACE(obj);                           \
99            }                                                            \
100       }                                                                 \
101       return intr;                                                      \
102   }                                                                     \
103   I##INTERFACE::I##INTERFACE() { }                                      \
104   I##INTERFACE::~I##INTERFACE() { }
```

So if we replace expand these two macros in IServiceManager.h & IServiceManager.cpp file with the appropriate replacement parameters they look like the following:

1. class ***IServiceManager*** : public IInterface
   {

```
public:
   static const android::String16 descriptor;
2.    static android::sp<IServiceManager> asInterface( const
android::sp<android::IBinder>& obj);
3.    virtual const android::String16& getInterfaceDescriptor() const;
4.    IServicemanager();
5.    virtual ~IServiceManager();
```

……..
……..
……
…..

And in
IServiceManager.cpp

```
1.
2.    const android::String16
IServiceManager::descriptor("android.os.IServiceManager");
3.    const android::String16&
4.          IServiceManager::getInterfaceDescriptor() const {
5.       return  IServiceManager::descriptor;
6.    }
7.    android::sp<IServiceManager> IServiceManager::asInterface(
8.          const android::sp<android::IBinder>& obj)
9.    {
10.      android::sp< IServiceManager> intr;
11.      if (obj != NULL) {
12.         intr = static_cast<IServiceManager*>(
13.            obj->queryLocalInterface(
14.                  IServiceManager::descriptor).get());
15.         if (intr == NULL) {
16.            intr = new BpServiceManager(obj);
17.         }
```

```
18.         }
19.         return intr;
20.     }
21.     IServiceManager::IServiceManager() { }
22.     IServiceManager::~IServiceManager { }
```

If you look at line 15, you will get how we get an handle to the BpServiceManager.

now once we get the reference of the Service Manager, we next call

```
public IBinder getService(String name) throws RemoteException {
116         Parcel data = Parcel.obtain();
117         Parcel reply = Parcel.obtain();
118         data.writeInterfaceToken(IServiceManager.descriptor);
119         data.writeString(name);
120         mRemote.transact(GET_SERVICE_TRANSACTION, data, reply, 0);
121         IBinder binder = reply.readStrongBinder();
122         reply.recycle();
123         data.recycle();
124         return binder;
125     }
```

from ServiceManagerNative.java. in this function we pass the service that we are looking for.

It returns the reference to the needed service through the function getService.

The getService function from /frameworks/base/libs/binder/IServiceManager.cpp looks like the following:

```
  virtual sp<IBinder> getService(const String16& name) const
134     {
```

```
135     unsigned n;
136     for (n = 0; n < 5; n++){
137         sp<IBinder> svc = checkService(name);
138         if (svc != NULL) return svc;
139         LOGI("Waiting for service %s...\n", String8(name).string());
140         sleep(1);
141     }
142     return NULL;
143
```

And the above **checkService(name)** looks like the following:

```
1.  virtual sp<IBinder> checkService( const String16& name) const
2.
3.      {
4.          Parcel data, reply;
5.
6.          data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());
7.
8.          data.writeString16(name);
9.
10.         remote()->transact(CHECK_SERVICE_TRANSACTION, data, &reply);
11.
12.         return reply.readStrongBinder();
13.
14.     }
```

So it actually calls a remote service and pass CHECK_SERVICE_TRANSACTION code (its an enum value of 2) to it.

This remote service is actually implemented in
frameworks/base/cmds/servicemanager/service_manager.c
and its onTransact looks like the following.

```
    switch(txn->code) {
            case SVC_MGR_GET_SERVICE:
```

```
            case SVC_MGR_CHECK_SERVICE:
a.              s = bio_get_string16(msg, &len);
b.              ptr = do_find_service(bs, s, len);
c.              if (!ptr)
d.                  break;
e.              bio_put_ref(reply, ptr);
f.              return 0;
```

Hence we end up calling the function named do_find_service which gets a reference to the service and returns it back.

The do_find_service from the same file looks as follows:

```
void *do_find_service(struct binder_state *bs, uint16_t *s, unsigned len)

{

    struct svcinfo *si;

    si = find_svc(s, len);



//    ALOGI("check_service('%s') ptr = %p\n", str8(s), si ? si->ptr : 0);

    if (si && si->ptr) {

        return si->ptr;

    } else {

        return 0;

    }

}
```

find_svc looks as follows:

```
struct svcinfo *find_svc(uint16_t *s16, unsigned len)

{

    struct svcinfo *si;



    for (si = svclist; si; si = si->next) {

        if ((len == si->len) &&

            !memcmp(s16, si->name, len * sizeof(uint16_t))) {

            return si;

        }

    }

    return 0;

}
```

As it becomes clear that it traverses through the svclist and returns the the service we are looking for.

Hope it helps the Android learners to know about the internal of Android services.