# 前言

四大核心组件之一的 Content Provider 同样需要在容器内单独做实现。我们主要考虑解决两个问题：

1. 容器内应用如何获取其他应用的 Content Provider
2. 容器如何为容器内的应用实现 Content Provider

很明显，直到 APP 被安装的那一刻，容器都不知道自己未来要实现怎么样的 Provider，因此我们需要考虑一种办法能够动态的安装这些组件。

# Content Provider 容器内实现

在之前启动 `Activity` 时有一个函数 `handleBindApplication` 用来绑定 Application 对象，而在这个函数中会调用 `installProviders` 来安装所有的 `Providers`：

```
private void installProviders(Context context, String processName,
List<ProviderInfo> provider) {
  long origId = Binder.clearCallingIdentity();
  try {
    for (ProviderInfo providerInfo : provider) {
      try {
        if (processName.equals(providerInfo.processName) ||
providerInfo.multiprocess) {
          Slog.d(TAG, "installProviders:" + providerInfo);
          installProvider(SandBoxCore.mainThread(), context, providerInfo,
null);
        }
      } catch (Throwable ignored) {
        ignored.printStackTrace();
      }
    }
  } finally {
    Binder.restoreCallingIdentity(origId);
    ContentProviderDelegate.init();
  }
}
```

其遍历了入参里的 `ProviderInfo` 数组，并调用 `installProvider` 来安装每个 `Provider`：

```
public static void installProvider(
    Object mainThread, Context context, ProviderInfo providerInfo, Object
```

```
holder)
    throws Throwable {
  Method installProvider =
      Reflector.findMethodByFirstName(mainThread.getClass(),
"installProvider");
  if (installProvider != null) {
    installProvider.setAccessible(true);
    installProvider.invoke(mainThread, context, holder, providerInfo, false,
true, true);
  }
}
```

这里就是直接调用原生的 `installProvider` 来完成安装了，并不需要有什么额外的操作。而且跟 Service 或 Activity 一样，VirtualApp 给它们提前占好了坑，用在 Blackbox 中是通过 ProxyContentProvider 来实现的：

```
<provider
  android:name=".proxy.ProxyContentProvider$P0"
  android:authorities="${applicationId}.proxy_content_provider_0"
  android:exported="true"
  android:process=":p0" />
```

这种描述其实有点问题，因为这些 ProxyContentProvider 其实并不是为了占坑而实现的，包括安装之类的操作其实都没有做相关的替换之类的操作。

然后在上面的操作完成以后，最后还有一个初始化的操作：

```
ContentProviderDelegate.init();
```

```
public static void init() {
  clearSettingProvider();

  SandBoxCore.getContext()
      .getContentResolver()
      .call(Uri.parse("content://settings"), "", null, null);
  Object activityThread = SandBoxCore.mainThread();
  ArrayMap<Object, Object> map =
      (ArrayMap<Object, Object>)
BRActivityThread.get(activityThread).mProviderMap();

  for (Object value : map.values()) {
    String[] mNames =
BRActivityThreadProviderClientRecordP.get(value).mNames();
```

```
    if (mNames == null || mNames.length <= 0) {
      continue;
    }
    String providerName = mNames[0];
    if (!sInjected.contains(providerName)) {
      sInjected.add(providerName);
      final IInterface iInterface =
BRActivityThreadProviderClientRecordP.get(value).mProvider();
      BRActivityThreadProviderClientRecordP.get(value)
          ._set_mProvider(
              new ContentProviderStub().wrapper(iInterface,
SandBoxCore.getHostPkg()));
      BRActivityThreadProviderClientRecordP.get(value)._set_mNames(new
String[] {providerName});
    }
  }
}
```

1. 在这里先获取了目标进程里的 `mProviderMap` ，这个对象记录了进程中所有的 contentProvider。
2. 遍历这个 Map，然后把里面的 mProvider 全部替换成 ContentProviderStub

注意到这个地方相当于把那些注册好了的 mProvider 全部包了一层：、

```
public IInterface wrapper(final IInterface contentProviderProxy, final String
appPkg) {
  mBase = contentProviderProxy;
  mAppPkg = appPkg;
  injectHook();
  // 这里返回的是 ContentProviderStub
  return (IInterface) getProxyInvocation();
}
```

然后对应的 `invoke` 函数：

```
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
  if ("asBinder".equals(method.getName())) {
    return method.invoke(mBase, args);
  }
  if (args != null && args.length > 0) {
    Object arg = args[0];
    if (arg instanceof String) {
```

```
        args[0] = mAppPkg;
      } else if
(arg.getClass().getName().equals(BRAttributionSource.getRealClass().getName())
) {
        ContextCompat.fixAttributionSourceState(arg, BActivityThread.getBUid());
      }
    }
    try {
      Object result = method.invoke(mBase, args);
      if (result instanceof Bundle) {
        Log.d(TAG, "result " + result);
      }
      return result;
    } catch (Throwable e) {
      throw e.getCause();
    }
}
```

相当于最终会得到一个 `ContentProviderStub` 来充当 `ContentProvider`，然后把其中的 invoke 做了点 hook。

而在那些需要使用 `ContentProvider` 的进程中，具体来说，VirtualApp 对那些需要使用 `ContentProvider` 的应用做了些手脚。在容器中，如果有哪个进程想要获取另外一个进程的 `ContentProvider`，就需要调用 `getContentProvider`：

```
@ProxyMethod("getContentProvider")
public static class GetContentProvider extends MethodHook {
  @Override
  protected Object hook(Object who, Method method, Object[] args) throws
Exception {
    int authIndex = getAuthIndex();
    Object auth = args[authIndex];
    Object content = null;

    if (auth instanceof String) {
      if (ProxyManifest.isProxy((String) auth)) {
        return method.invoke(who, args);
      }

      if (BuildCompat.isQ()) {
        args[1] = SandBoxCore.getHostPkg();
      }

      if (auth.equals("settings")
          || auth.equals("media")
```

```
            || auth.equals("telephony")
            || auth.equals("com.huawei.android.launcher.settings")
            || auth.equals("com.hihonor.android.launcher.settings")) {
          content = method.invoke(who, args);
          ContentProviderDelegate.update(content, (String) auth);
          return content;
        } else {
          Log.d(TAG, "hook getContentProvider: " + auth);

          ProviderInfo providerInfo =
              SandBoxCore.getBPackageManager()
                  .resolveContentProvider(
                      (String) auth, GET_META_DATA,
BActivityThread.getUserId());
          if (providerInfo == null) {
            return null;
          }

          Log.d(TAG, "hook app: " + auth);
          IBinder providerBinder = null;
          if (BActivityThread.getAppPid() != -1) {
            AppConfig appConfig =
                SandBoxCore.getBActivityManager()
                    .initProcess(
                        providerInfo.packageName,
                        providerInfo.processName,
                        BActivityThread.getUserId());
            if (appConfig.bpid != BActivityThread.getAppPid()) {
              providerBinder =

SandBoxCore.getBActivityManager().acquireContentProviderClient(providerInfo);
            }
            args[authIndex] = ProxyManifest.getProxyAuthorities(appConfig.bpid);
            args[getUserIndex()] = SandBoxCore.getHostUserId();
          }
          if (providerBinder == null) return null;

          content = method.invoke(who, args);
          Reflector.with(content).field("info").set(providerInfo);
          Reflector.with(content)
              .field("provider")
              .set(
                  new ContentProviderStub()
                      .wrapper(

BRContentProviderNative.get().asInterface(providerBinder),
```

```
                        BActivityThread.getAppPackageName()));
            }

            return content;
        }
        return method.invoke(who, args);
    }

    private int getAuthIndex() {
        // 10.0
        if (BuildCompat.isQ()) {
            return 2;
        } else {
            return 1;
        }
    }

    private int getUserIndex() {
        return getAuthIndex() + 1;
    }
}
```

1. 首先，尝试从 `BPackageManager` 拿到 `ProviderInfo` 。
2. 如果拿到了，会尝试调用 `initProcess` 把目标进程唤起
3. 通过 `acquireContentProviderClient` 来得到原先注册的那个 `providerBinder`
4. 用 `ProxyManifest.getProxyAuthorities` 把入参替换成 `"%s.proxy_content_provider_%d"`
5. 最后去掉原生的那个函数去获取目标，这里应该会返回一个 ProxyContentProvider，不过它本来也是继承自 ContentProvider 的类，其实差不多
6. 修改 `info` 为 `ProviderInfo` ，修改 `provider` 为 `providerBinder` 从而将它伪造成真正 `ContentProvider`
7. 返回伪造后的结果

```
    Reflector.with(content)
            .field("provider")
            .set(
                new ContentProviderStub()
                    .wrapper(
                        BRContentProviderNative.get().asInterface(providerBinder),
                        BActivityThread.getAppPackageName()));
}
```

最后这里替换 `provider` 为 `providerBinder` 的时候有一层 `ContentProviderStub` 的包装。不过在调用它的 invoke 函数的时候会使用传入的 providerBinder 进行调用，因此没有问题。

```
Object result = method.invoke(mBase, args);
```

综上所述，VirtualApp 实现了在容器内伪造 Content Provider 的能力。

## 参考文章


https://gityuan.com/2016/07/30/content-provider/
https://blog.csdn.net/ganyao939543405/article/details/76253562
https://juejin.cn/post/7028124957141893150