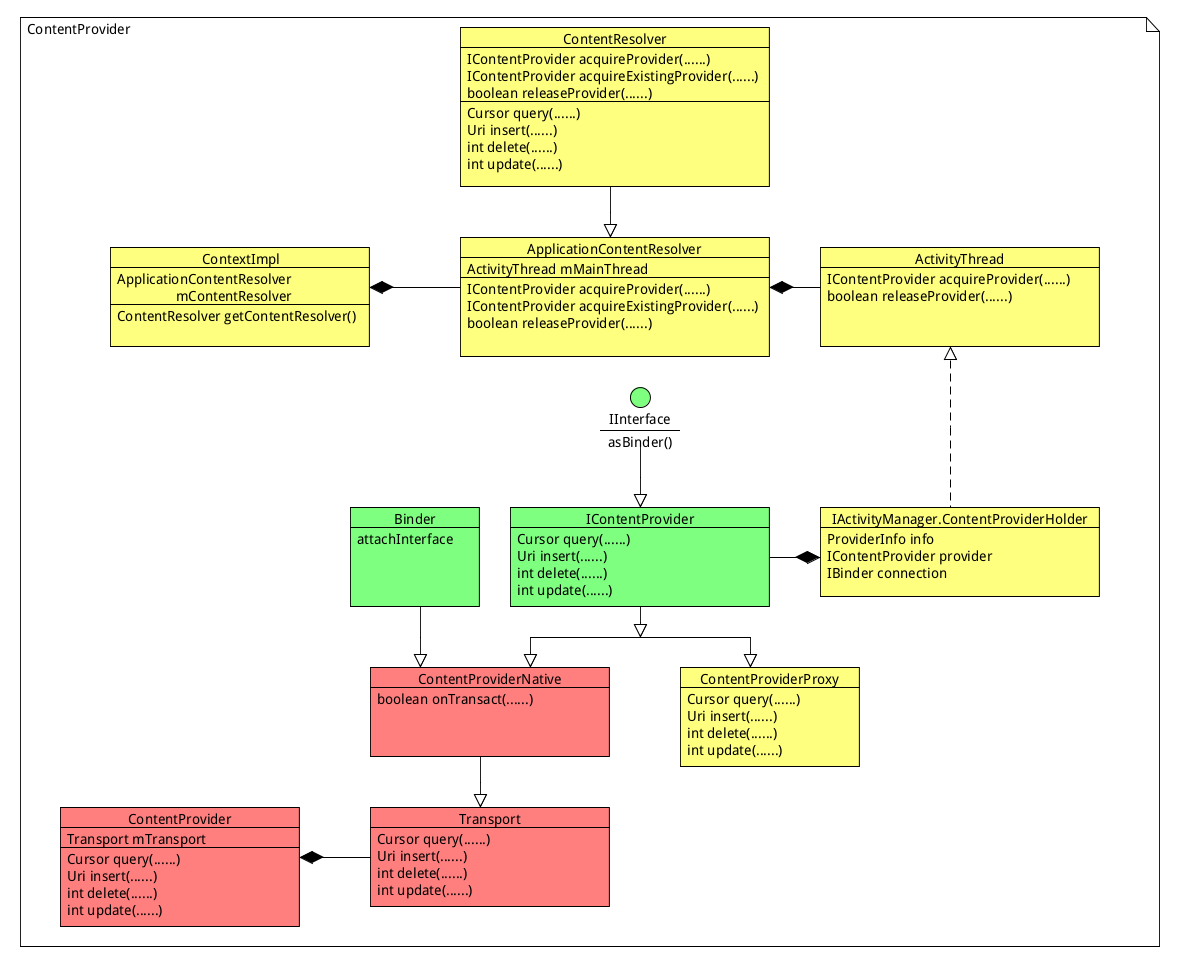
**ContentProvider**

1. ContentProvider组成
2. 使用Content provider的client端的行为

1@调用acquireExistingProvider函数检查本地是否有活着的IContentProvider bp对象：

public final IContentProvider acquireExistingProvider(

Context c, String auth, int userId, boolean stable) {

synchronized (mProviderMap) {

final ProviderKey key = new ProviderKey(auth, userId);

**//通过auth获取mProviderMap是否已经存在IContentProvider**

final ProviderClientRecord pr = mProviderMap.get(key);

if (pr == null) {

return null;

}

IContentProvider provider = pr.mProvider;

IBinder jBinder = provider.asBinder();

**//当存在的时候获取ContentProviderProxy来检查BN端是否运行**

if (!jBinder.isBinderAlive()) {

handleUnstableProviderDiedLocked(jBinder, true);

return null;

}

ProviderRefCount prc = mProviderRefCountMap.get(jBinder);

if (prc != null) {

incProviderRefLocked(prc, stable);

}

**//当满足以上两个条件直接返回IContentProvider 对象否则返回null**

return provider;

}

}

2@调用到AMS的getContentProviderImpl函数来获取ContentProvider bp对象：

ProviderInfo cpi = null;

try {

checkTime(startTime, "getContentProviderImpl: before resolveContentProvider");

**//通过auth字符串在PackageManager中的mProvidersByAuthority获取**

**//ProviderInfo 对象**

cpi = AppGlobals.getPackageManager().

resolveContentProvider(name,

STOCK\_PM\_FLAGS | PackageManager.GET\_URI\_PERMISSION\_PATTERNS, userId);

checkTime(startTime, "getContentProviderImpl: after resolveContentProvider");

} catch (RemoteException ex) {

}

if (cpi == null) {

return null;

}

ContentProviderRecord cpr;

ComponentName comp = new ComponentName(cpi.packageName, cpi.name);

**//当该contentprovider不在mProviderMap容器中的时候通过packageManager**

**//获取ApplicationInfo 来创建相应的ContentProviderRecord**

cpr = mProviderMap.getProviderByClass(comp, userId);

checkTime(startTime, "getContentProviderImpl: after getProviderByClass");

final boolean firstClass = cpr == null;

if (firstClass) {

final long ident = Binder.clearCallingIdentity();

try {

ApplicationInfo ai =

AppGlobals.getPackageManager().

getApplicationInfo(

cpi.applicationInfo.packageName,

STOCK\_PM\_FLAGS, userId);

if (ai == null) {

return null;

}

ai = getAppInfoForUser(ai, userId);

cpr = new ContentProviderRecord(this, cpi, ai, comp, singleton);

}

}

final int N = mLaunchingProviders.size();

int i;

**//检查contentprovider进程是否已经请求发起启动**

for (i = 0; i < N; i++) {

if (mLaunchingProviders.get(i) == cpr) {

break;

}

}

**//没有发起启动调用startProcessLocked函数启动进程**

if (i >= N) {

AppGlobals.getPackageManager().setPackageStoppedState(

cpr.appInfo.packageName, false, userId);

ProcessRecord proc = getProcessRecordLocked(

cpi.processName, cpr.appInfo.uid, false);

if (proc != null && proc.thread != null) {

if (!proc.pubProviders.containsKey(cpi.name)) {

proc.pubProviders.put(cpi.name, cpr);

proc.thread.scheduleInstallProvider(cpi);

}

} else {

proc = startProcessLocked(cpi.processName,

cpr.appInfo, false, 0, "content provider",

new ComponentName(cpi.applicationInfo.packageName,

cpi.name), false, false, false);

if (proc == null) {

return null;

}

}

cpr.launchingApp = proc;

mLaunchingProviders.add(cpr);

}

}

if (firstClass) {

mProviderMap.putProviderByClass(comp, cpr);

}

mProviderMap.putProviderByName(name, cpr);

conn = incProviderCountLocked(r, cpr, token, stable);

if (conn != null) {

conn.waiting = true;

}

**//等待contentprovider进程publishContentProviders**

synchronized (cpr) {

while (cpr.provider == null) {

if (cpr.launchingApp == null) {

return null;

}

try {

if (conn != null) {

conn.waiting = true;

}

cpr.wait();

} catch (InterruptedException ex) {

} finally {

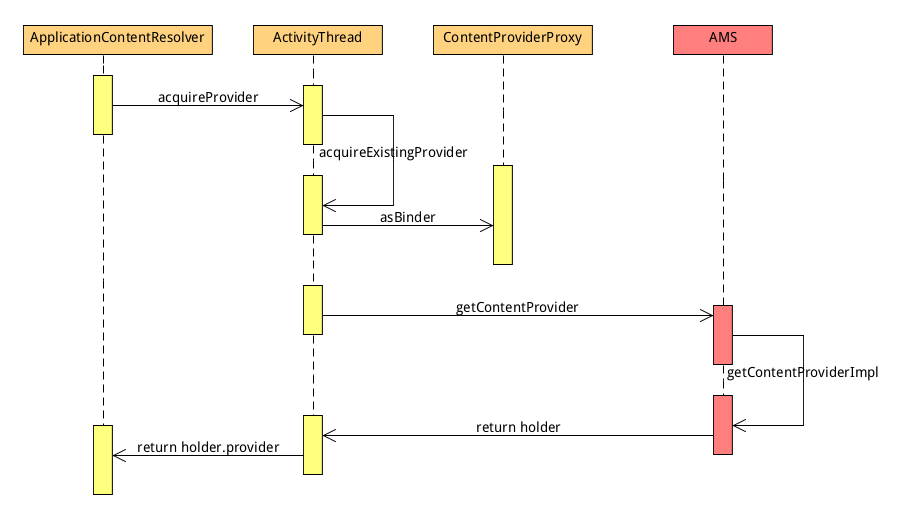
if (conn != null) {

conn.waiting = false;

}

}

}



3.包含contentProvider的服务端进程：

在contentprovider所属进程主线程调用handleBindApplication函数中调用installContentProviders函数来进行publish content provider

private void installContentProviders(

Context context, List<ProviderInfo> providers) {

final ArrayList<IActivityManager.ContentProviderHolder> results =

new ArrayList<IActivityManager.ContentProviderHolder>();

**//创建本地的ContentProvider对象同时使用getIContentProvider**

**//接口来获取IContentProvider对象，也就是Transport对象**

**//使用ContentProvider对象创建ContentProviderHolderI来给AMS**

for (ProviderInfo cpi : providers) {

IActivityManager.ContentProviderHolder cph = installProvider(context, null, cpi,

false /\*noisy\*/, true /\*noReleaseNeeded\*/, true /\*stable\*/);

if (cph != null) {

cph.noReleaseNeeded = true;

results.add(cph);

}

}

}

**//通知AMS contentprovier进程启动完成**

try {

ActivityManagerNative.getDefault().publishContentProviders(

getApplicationThread(), results);

}

}

private IActivityManager.ContentProviderHolder installProvider(Context context,

IActivityManager.ContentProviderHolder holder, ProviderInfo info,

boolean noisy, boolean noReleaseNeeded, boolean stable) {

ContentProvider localProvider = null;

IContentProvider provider;

if (holder == null || holder.provider == null) {

**//创建ContentProvider 和IContentProvider 对象。**

final java.lang.ClassLoader cl = c.getClassLoader();

localProvider = (ContentProvider)cl.

loadClass(info.name).newInstance();

provider = localProvider.getIContentProvider();

if (provider == null) {

return null;

}

localProvider.attachInfo(c, info);

}

IActivityManager.ContentProviderHolder retHolder;

synchronized (mProviderMap) {

IBinder jBinder = provider.asBinder();

if (localProvider != null) {

ComponentName cname = new ComponentName(info.packageName, info.name);

ProviderClientRecord pr = mLocalProvidersByName.get(cname);

**//使用IContentProvider 对象创建ContentProviderHolder 给AMS**

holder = new IActivityManager.ContentProviderHolder(info);

holder.provider = provider;

holder.noReleaseNeeded = true;

pr = installProviderAuthoritiesLocked(provider, localProvider, holder);

mLocalProviders.put(jBinder, pr);

mLocalProvidersByName.put(cname, pr);

retHolder = pr.mHolder;

}

}

return retHolder;

}

public IContentProvider getIContentProvider() {

return mTransport;

}

private void attachInfo(Context context, ProviderInfo info, boolean testing) {

if (mContext == null) {

mContext = context;

ContentProvider.this.onCreate();

}

}

public final void publishContentProviders(IApplicationThread caller,

List<ContentProviderHolder> providers) {

final int N = providers.size();

for (int i = 0; i < N; i++) {

ContentProviderHolder src = providers.get(i);

ContentProviderRecord dst = r.pubProviders.get(src.info.name);

if (dst != null) {

ComponentName comp = new ComponentName(dst.info.packageName, dst.info.name);

mProviderMap.putProviderByClass(comp, dst);

String names[] = dst.info.authority.split(";");

for (int j = 0; j < names.length; j++) {

mProviderMap.putProviderByName(names[j], dst);

}

int launchingCount = mLaunchingProviders.size();

int j;

boolean wasInLaunchingProviders = false;

for (j = 0; j < launchingCount; j++) {

if (mLaunchingProviders.get(j) == dst) {

mLaunchingProviders.remove(j);

wasInLaunchingProviders = true;

j--;

launchingCount--;

}

}

if (wasInLaunchingProviders) {

mHandler.removeMessages(CONTENT\_PROVIDER\_PUBLISH\_TIMEOUT\_MSG, r);

}

synchronized (dst) {

dst.provider = src.provider;

dst.proc = r;

**//唤醒在getContentProviderImpl函数创建的ContentProviderRecord对象等**

**//待的线程**

dst.notifyAll();

}

updateOomAdjLocked(r);

maybeUpdateProviderUsageStatsLocked(r, src.info.packageName,

src.info.authority);

}

}

}

}

3.releaseProvider