Xposed框架中XSharePreference的使用

原创

Fly20141201 于 2018-07-25 22:47:47 发布 5718 收藏 6
分类专栏： Android逆向学习 Android系统安全和逆向分析研究 文章标签： xposed XSharePreference hook

本文链接：https://blog.csdn.net/QQ1084283172/article/details/81194406


Android逆向学习 同时被 2 个专栏收录
58 篇文章 6 订阅
订阅专栏


Android系统安全和逆向分析研究
72 篇文章 59 订阅
订阅专栏

本文博客地址：https://blog.csdn.net/QQ1084283172/article/details/81194406

在Xposed框架的模块编写中，通常希望我们自己写的Android界面控制部分能够对Xposde框架的模块进行配置，也就说在当前Android应用中写的SharePreference文件，在其他Android应用加载该Xposde模块时，在执行的Xposed模块代码中能够访问到前面提到的SharePreference配置文件。当然了将配置文件写在Android应用的sd卡路径下是可以的，但是到了Android系统7.0的高版本时，基于sd卡文件的访问权限问题，使得这种方法不是很好使，当然了基于SharePreference配置文件的数据共享方法也不好使。这里只是学习一下Xposed框架中XSharePreference的使用方法，具体遇到的问题就是《一个基于xposed和inline hook的一代壳脱壳工具》中，关于Xposed模块共享配置文件问题的讨论，总之呢，到了Android 7.0版本的系统，基于权限和安全性的考虑，这两种方法都不是最佳的共享数据的通用方法。

一、普通情况下，Android应用跨进程共享文件数据的方法，可以参考博客《读、写其他应用程序的SharedPreferences》的讲解。

要读、写其他Android应用的SharedPreferences文件，需要共享该SharedPreferences文件的Android应用程序指定相应的访问权限，例如：MODE_WORLD_READABLE表明该SharedPreferences可被其他应用程序读取；指定MODE_WORLD_WRITEABLE，表明该SharedPreferences可被其他程序写入。
操作步骤：
1、创建其他Android应用程序对应的Context。
2、调用上面Context的getSharedPreferences(String name,int mode)，获取相应的SharedPreferences对象。
3、如果需要写入数据，调用SharedPreferences的edit()方法获取相应的Editor即可。

摘抄一下作者冯健-developer大大的博客代码，进行演示说明：

共享SharedPreferences配置文件数据的Android应用：

```java
package com.lovo.usecount;

import android.os.Bundle;
import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.widget.TextView;

public class UseCount extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                TextView show = (TextView) findViewById(R.id.main_tv);
                SharedPreferences sp = getSharedPreferences("count",
                                MODE_WORLD_READABLE);
                // 读取SharedPreferences里的count数据
                int count = sp.getInt("count", 0);
                // 显示程序以前使用的次数
                show.setText("该程序之前被使用了" + count + "次");
                Editor editor = sp.edit();
                // 存入数据
                editor.putInt("count", ++count);
                // 提交修改
                editor.commit();
        }

}
```

共享SharedPreferences配置文件数据的Android应用

获取共享的SharedPreferences配置文件数据的Android应用：

```java
package com.lovo;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.widget.TextView;                    获取共享的SharedPreferences配置文件数据的应用

public class ReadOtherPreferences extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_preferences_test);
                Context useCount = null;
                try {
                        // 获取其他程序对应的Context
                        useCount = createPackageContext("com.lovo.usecount",
                                        Context.CONTEXT_IGNORE_SECURITY);
                } catch (NameNotFoundException e) {
                        e.printStackTrace();
                }
                // 使用其他程序的COntext获取对应的SharedPreferences
                SharedPreferences ps = useCount.getSharedPreferences("count",
                                Context.MODE_WORLD_READABLE);
                // 读取数据
                int count = ps.getInt("count", 0);
                TextView show = (TextView) findViewById(R.id.activity_preferences_test_tv_show);
                // 显示读取的数据内容
                show.setText("UseCount应用程序之前被使用了" + count + "次");
        }
}                                                  https://blog.csdn.net/QQ1084283172
```

但是这个Android跨进程共享SharedPreferences文件数据的方案，在Android的高版本系统上并不好使，具体的可以参考博客《SharedPreferences多进程解决方案》的描述，google官方还是推荐使用ContentProvider的方法进行替代，具体的内容如下：

刚才笔者无聊（其实想自己验证一下，尝试了SharedPreferences其他mode,效果惨不忍睹，{MODE_WORLD_READABLE}和{MODE_WORLD_WRITEABLE}跨进程也有一堆问题，所以想跨进程用SharedPreferences，尽早脱坑，而且在Android N版本，系统会直接报错的）

```
    private void checkMode(int mode) {
        if (getApplicationInfo().targetSdkVersion >= Build.VERSION_CODES.N) {
            if ((mode & MODE_WORLD_READABLE) != 0) {
                throw new SecurityException("MODE_WORLD_READABLE no longer supported");
            }
            if ((mode & MODE_WORLD_WRITEABLE) != 0) {
                throw new SecurityException("MODE_WORLD_WRITEABLE no longer supported");
            }
        }
    }
```

现在同一个app内部多进程也越来越常见了，所以这个问题还是需要针对性解决的，按google的想法，是希望通过Content Provider去set,query数据，ContentProvider内部对这种多进程做了处理，这样我们就不用去跨进程通信这一块，将我们的核心思想放在数据存取上了。

参考过百度上搜索出来的SharedPreferences多进程解决方案，比较多的都是采用ContentProvider,然后封装一下数据库的操作。其实很多情况下，我们所需要使用的功能并不是那么复杂，根本目的就是想跨进程的情况下还能使用，而且只使用ContentProvider+数据库，个人感觉还是太重了，一般使用SharedPreferences，都是简单的键值对，数据类型本来就不会太复杂的，用数据库岂不是大材小用了。

所以设想也很简单，这个解决方案：
- 这个类继承于ContentProvider，让google内部帮我们实现多进程通信机制，省了一个大麻烦
- 希望使用的人（程序员），还是当以往使用SharedPreferences的感觉去操作，我意思是implements SharedPreferences这个接口，把里面的Editor的putString()，commit()等方法实现好，这样程序员就不用学习这个SharedPreferences究竟要怎么用了。以前该干嘛就干嘛

二、Xposed框架下，Android应用与编写的Xposed模块进行SharedPreferences文件数据共享的实现。

对于Android应用与编写的Xposed模块进行SharedPreferences文件数据的共享，Xposed框架提供了XSharePreference的解决方法，不过需要说明的Xposed框架提供的XSharePreference只能实现SharedPreferences文件数据共享的读，不支持跨进程SharedPreferences文件数据共享的写。SharePrefernce实际上是对应路径/data/data/<Package Name>/shared_prefs/<Package Name>_preferences.xml的文件，里面按键值对的形式存放。XSharedPreferences做的工作实际上就是根据<Package Name>自己去获取和解析这个配置文件，然后读出该配置文件中的数据存放在一个HashMap当中，这些可以从XSharedPreferences的源码中了解到。

```
package de.robv.android.xposed;

import android.annotation.SuppressLint;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Environment;
import android.preference.PreferenceManager;
import android.util.Log;

import com.android.internal.util.XmlUtils;

import org.xmlpull.v1.XmlPullParserException;

import java.io.File;
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import de.robv.android.xposed.services.FileResult;

/**
 * This class is basically the same as SharedPreferencesImpl from AOSP, but
 * read-only and without listeners support. Instead, it is made to be
 * compatible with all ROMs.
 */
public final class XSharedPreferences implements SharedPreferences {
 private static final String TAG = "XSharedPreferences";
 private final File mFile;
 private final String mFilename;
 private Map<String, Object> mMap;
 private boolean mLoaded = false;
 private long mLastModified;
 private long mFileSize;

 /**
  * Read settings from the specified file.
  * @param prefFile The file to read the preferences from.
  */
 public XSharedPreferences(File prefFile) {
  mFile = prefFile;
  mFilename = mFile.getAbsolutePath();
  startLoadFromDisk();
 }

 /**
  * Read settings from the default preferences for a package.
  * These preferences are returned by {@link PreferenceManager#getDefaultSharedPreferences}.
  * @param packageName The package name.
  */
 public XSharedPreferences(String packageName) {

  // 调用XSharedPreferences(String packageName, String prefFileName)函数
  this(packageName, packageName + "_preferences");
 }

 /**
  * Read settings from a custom preferences file for a package.
  * These preferences are returned by {@link Context#getSharedPreferences(String, int)}.
  * @param packageName The package name.
  * @param prefFileName The file name without ".xml".
  */
 public XSharedPreferences(String packageName, String prefFileName) {

  // Environment.getDataDirectory()获取Android系统的/data目录
  // 在/data/data/路径下创建共享xml文件
  mFile = new File(Environment.getDataDirectory(), "data/" + packageName + "/shared_prefs/" + prefFileName
  // 获取文件全路径
  mFilename = mFile.getAbsolutePath();
  startLoadFromDisk();
 }
```

```
/**
 * Tries to make the preferences file world-readable.
 *
 * <p><strong>Warning:</strong> This is only meant to work around permission "fix" functions that are part
 * of some recoveries. It doesn't replace the need to open preferences with {@code MODE_WORLD_READABLE}
 * in the module's UI code. Otherwise, Android will set stricter permissions again during the next save.
 *
 * <p>This will only work if executed as root (e.g. {@code initZygote()}) and only if SELinux is disabled.
 *
 * @return {@code true} in case the file could be made world-readable.
 */
@SuppressLint("SetWorldReadable")
public boolean makeWorldReadable() {
 if (!SELinuxHelper.getAppDataFileService().hasDirectFileAccess())
  return false; // It doesn't make much sense to make the file readable if we wouldn't be able to access i

 if (!mFile.exists()) // Just in case - the file should never be created if it doesn't exist.
  return false;
 // 设置文件为全局可读
 return mFile.setReadable(true, false);
}

/**
 * Returns the file that is backing these preferences.
 *
 * <p><strong>Warning:</strong> The file might not be accessible directly.
 */
public File getFile() {
 return mFile;
}

private void startLoadFromDisk() {
 synchronized (this) {
  mLoaded = false;
 }
 new Thread("XSharedPreferences-load") {
  @Override
  public void run() {
   synchronized (XSharedPreferences.this) {
    loadFromDiskLocked();
   }
  }
 }.start();
}

@SuppressWarnings({ "rawtypes", "unchecked" })
private void loadFromDiskLocked() {
 if (mLoaded) {
  return;
 }

 Map map = null;
 FileResult result = null;
 try {
  result = SELinuxHelper.getAppDataFileService().getFileInputStream(mFilename, mFileSize, mLastModified);
  if (result.stream != null) {
   map = XmlUtils.readMapXml(result.stream);
   result.stream.close();
  } else {
```

```
    // The file is unchanged, keep the current values
    map = mMap;
   }
  } catch (XmlPullParserException e) {
   Log.w(TAG, "getSharedPreferences", e);
  } catch (FileNotFoundException ignored) {
   // SharedPreferencesImpl has a canRead() check, so it doesn't log anything in case the file doesn't exis
  } catch (IOException e) {
   Log.w(TAG, "getSharedPreferences", e);
  } finally {
   if (result != null && result.stream != null) {
    try {
     result.stream.close();
    } catch (RuntimeException rethrown) {
     throw rethrown;
    } catch (Exception ignored) {
    }
   }
  }

  mLoaded = true;
  if (map != null) {
   mMap = map;
   mLastModified = result.mtime;
   mFileSize = result.size;
  } else {
   mMap = new HashMap<>();
  }
  notifyAll();
 }

 /**
  * Reload the settings from file if they have changed.
  *
  * <p><strong>Warning:</strong> With enforcing SELinux, this call might be quite expensive.
  */
 public synchronized void reload() {
  if (hasFileChanged())
   startLoadFromDisk();
 }

 /**
  * Check whether the file has changed since the last time it has been loaded.
  *
  * <p><strong>Warning:</strong> With enforcing SELinux, this call might be quite expensive.
  */
 public synchronized boolean hasFileChanged() {
  try {
   FileResult result = SELinuxHelper.getAppDataFileService().statFile(mFilename);
   return mLastModified != result.mtime || mFileSize != result.size;
  } catch (FileNotFoundException ignored) {
   // SharedPreferencesImpl doesn't log anything in case the file doesn't exist
   return true;
  } catch (IOException e) {
   Log.w(TAG, "hasFileChanged", e);
   return true;
  }
 }

 private void awaitLoadedLocked() {
```

```
 while (!mLoaded) {
  try {
   wait();
  } catch (InterruptedException unused) {
  }
 }
}

/** @hide */
@Override
public Map<String, ?> getAll() {
 synchronized (this) {
  awaitLoadedLocked();
  return new HashMap<>(mMap);
 }
}

/** @hide */
@Override
public String getString(String key, String defValue) {
 synchronized (this) {
  awaitLoadedLocked();
  String v = (String)mMap.get(key);
  return v != null ? v : defValue;
 }
}

/** @hide */
@Override
@SuppressWarnings("unchecked")
public Set<String> getStringSet(String key, Set<String> defValues) {
 synchronized (this) {
  awaitLoadedLocked();
  Set<String> v = (Set<String>) mMap.get(key);
  return v != null ? v : defValues;
 }
}

/** @hide */
@Override
public int getInt(String key, int defValue) {
 synchronized (this) {
  awaitLoadedLocked();
  Integer v = (Integer)mMap.get(key);
  return v != null ? v : defValue;
 }
}

/** @hide */
@Override
public long getLong(String key, long defValue) {
 synchronized (this) {
  awaitLoadedLocked();
  Long v = (Long)mMap.get(key);
  return v != null ? v : defValue;
 }
}

/** @hide */
```

```
@Override
public float getFloat(String key, float defValue) {
 synchronized (this) {
  awaitLoadedLocked();
  Float v = (Float)mMap.get(key);
  return v != null ? v : defValue;
 }
}

/** @hide */
@Override
public boolean getBoolean(String key, boolean defValue) {
 synchronized (this) {
  awaitLoadedLocked();
  Boolean v = (Boolean)mMap.get(key);
  return v != null ? v : defValue;
 }
}

/** @hide */
@Override
public boolean contains(String key) {
 synchronized (this) {
  awaitLoadedLocked();
  return mMap.containsKey(key);
 }
}

/** @deprecated Not supported by this implementation. */
@Deprecated
@Override
public Editor edit() {
       // 不支持共享配置文件的写操作
 throw new UnsupportedOperationException("read-only implementation");
}

/** @deprecated Not supported by this implementation. */
@Deprecated
@Override
public void registerOnSharedPreferenceChangeListener(OnSharedPreferenceChangeListener listener) {
 throw new UnsupportedOperationException("listeners are not supported in this implementation");
}

/** @deprecated Not supported by this implementation. */
@Deprecated
@Override
public void unregisterOnSharedPreferenceChangeListener(OnSharedPreferenceChangeListener listener) {
 throw new UnsupportedOperationException("listeners are not supported in this implementation");
}

}
```

XSharedPreferences的api使用说明：https://api.xposed.info/reference/de/robv/android/xposed/XSharedPreferences.html

1.使用 XSharedPreferences(String packageName) 获取共享配置文件SharedPreferences的数据。

构造函数的源码如下，在构建XSharedPreferences对象设置默认的xml文件的文件名称为**packageName+"_preferences"**，很显然构造函数XSharedPreferences(String packageName)最终是调用的构造函数XSharedPreferences(String packageName, String prefFileName)来实现的，只是传给该构造函数的第2个参数是默认的packageName+"_preferences"。

```
/**
 * Read settings from the default preferences for a package.
 * These preferences are returned by {@link PreferenceManager#getDefaultSharedPreferences}.
 * @param packageName The package name.
 */
public XSharedPreferences(String packageName) {

    // 调用XSharedPreferences(String packageName, String prefFileName)函数
    this(packageName, prefFileName: packageName + "_preferences");
}

/**
 * Read settings from a custom preferences file for a package.
 * These preferences are returned by {@link Context#getSharedPreferences(String, int)}.
 * @param packageName The package name.
 * @param prefFileName The file name without ".xml".
 */
public XSharedPreferences(String packageName, String prefFileName) {

    // Environment.getDataDirectory()获取Android系统的/data目录
    // 在/data/data/路径下创建共享xml文件
    mFile = new File(Environment.getDataDirectory(), name: "data/" + packageName + "/shared_prefs/" + prefFileName + ".xml");
    // 获取文件全路径
    mFilename = mFile.getAbsolutePath();
    startLoadFromDisk();
}
```


XSharedPreferences(String packageName)函数的使用说明：

public XSharedPreferences (String packageName)

Read settings from the default preferences for a package. These preferences are returned by `PreferenceManager.getDefaultSharedPreferences(Context)`.

Parameters

packageName The package name.

保存数据到共享配置文件SharedPreferences中，使用的场景是自己写的Xposed框架一般都会做个界面，继承Activity的，保存数据是直接正常保存，唯一注意的就是初始化SharedPreferences的时候，权限要写成：**Activity.MODE_WORLD_READABLE**。这样的话，在其他的Xposed模块才能访问到这个共享的配置文件SharedPreferences，但是在Android 7.0的系统上还是会存在问题，不好使。

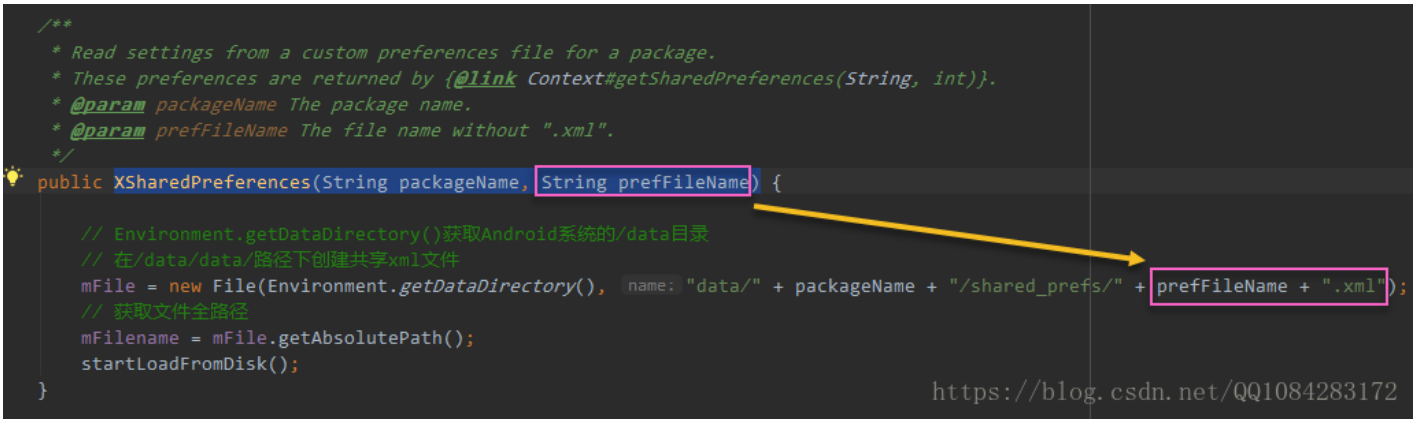
```
// 创建共享配置文件的时，设置为Activity.MODE_WORLD_READABLE模式
SharedPreferences sp = getActivity()
          .getSharedPreferences(getPackageName() + "_preferences", Activity.MODE_WORLD_READABLE);
SharedPreferences.Editor editor  = sp.edit();
// 保存数据到共享配置文件中
editor.putString("test_put", "test_put").commit();
```

在编写的Xposed模块中，对共享的SharedPreferences文件数据进行访问：

```
// 其中"com.xxx.yyy"为共享SharedPreferences文件数据的Android应用的包名
XSharedPreferences intance = new XSharedPreferences("xxx.yyy.zzz");
String strHello = intance.getString("test_put", "");
```

2.使用XSharedPreferences(String packageName, String prefFileName)获取共享配置文件SharedPreferences的数据。

构造函数的源码如下：



构造函数的使用说明：

public XSharedPreferences (String packageName, String prefFileName)

Read settings from a custom preferences file for a package. These preferences are returned by `Context.getSharedPreferences(String, int)`.

Parameters

packageName	The package name.
prefFileName	The file name without ".xml".

数据的保存：

```
SharedPreferences sp = getActivity().getSharedPreferences("config", Activity.MODE_WORLD_READABLE);
SharedPreferences.Editor editor = sp.edit();
editor.putBoolean("isOk", true).commit();
```

数据的访问：

在XSharePreference初始化完成后，一定要设置为可读取即makeWorldReadable()。XSharedPreferences是只读的，并不允许进行写操作，从源码中实现中就可以看出来，代码来自于博客《Xposed中的XSharePreference的使用》。

```
public class PreferenceUtils {

    public static final String AUTO_ALL ="auto_all";
    public static final String REPLY_CONTENT ="reply_content";
    public static final String REPLY_ORDER ="reply_order";
    public static final String REPLY_ROBOT ="reply_robot";
    public static final String REPLY_DESCRIBE ="reply_describe";
    private static XSharedPreferences intance = null;

    public static XSharedPreferences getIntance(){
        if (intance == null){
            intance = new XSharedPreferences("com.ydscience.fakemomo","config");
            intance.makeWorldReadable();
        }else {
            intance.reload();
        }
        return intance;
    }

    public static boolean openAll(){
        return getIntance().getBoolean(AUTO_ALL,false);
  }

    public static String replyContent(){
        return getIntance().getString(REPLY_CONTENT,"请设置自动回复内容");
    }
}
```

提供一个Xposed框架中XSharePreference使用的例子，来源于看雪论坛《一个基于xposed和inline hook的一代壳脱壳工具》下面关于该脱壳插件改进讨论的例子代码，在这个基础上根据需求进行修改即可：

```
/**
 * Created by virjar on 2018/2/24.<br>
 * 原方案，在宿主apk无sd卡权限时，无法hook无法读取配置，所以使用XSharePreference来处理多apk配置通信问题，这是因为shar
 */

class XSharePreferenceConfig {

    private static final String pluginPackage = "com.smartdone.dexdump";
    private static final String configName = "dumpapps";

    //可写的XSharedPreferences对象，这个可以使用在插件自己的apk里面使用
    private SharedPreferences writeXSharedPreferences = null;
    //只读的XSharedPreferences,这个可以让插件在宿主Android应用里面访问
    private SharedPreferences readXSharedPreferences = null;

    @SuppressLint("WorldReadableFiles")
    void initWriteSharedPreference(Context pluginHostContext) {
        if (!StringUtils.equals(pluginHostContext.getApplicationInfo().packageName, pluginPackage)) {
            throw new IllegalStateException("can not create write sharedPreference in host app");
        }
        if (writeXSharedPreferences != null) {
            return;
        }
        writeXSharedPreferences = pluginHostContext.getSharedPreferences(configName, Context.MODE_WORLD_REA
    }
```

```
    void initReadSharePreference() {
        if (readXSharedPreferences != null) {
            return;
        }
        readXSharedPreferences = new XSharedPreferences(pluginPackage, configName);
    }

    private SharedPreferences getBindSharePreference() {
        if (writeXSharedPreferences != null) {
            return writeXSharedPreferences;
        }
        if (readXSharedPreferences != null) {
            return readXSharedPreferences;
        }
        throw new IllegalStateException("not inited");
    }

    // 向共享配置文件中添加数据
    void addOne(String name) {
        SharedPreferences bindSharePreference = getBindSharePreference();
        Set<String> stringSet = Sets.newHashSet(Splitter.on(",").splitToList(bindSharePreference.getString(
        stringSet.add(name);
        bindSharePreference.edit().putString(configName, Joiner.on(",").join(stringSet)).apply();
    }

    // 从共享配置文件中移除数据
    void removeOne(String name) {
        SharedPreferences bindSharePreference = getBindSharePreference();
        Set<String> stringSet = Sets.newHashSet(Splitter.on(",").splitToList(bindSharePreference.getString(
        stringSet.remove(name);
        bindSharePreference.edit().putString(configName, Joiner.on(",").join(stringSet)).apply();
    }

    // 获取共享的配置文件中的数据
    Set<String> getConfig() {
        return Sets.newHashSet(Splitter.on(",").splitToList(getBindSharePreference().getString(configName,
    }

    static boolean contains(Set<String> lists, String name) {
        if (lists == null) {
            return false;
        }
        for (String l : lists) {
            if (l.equals(name)) {
                return true;
            }
        }
        return false;
    }

}
```

比较不错的博客参考链接，很多知识点都是结合Xposed的api参考文档和源码以及这些博客搬运过来：

Xposed的api参考文档: https://api.xposed.info/reference/packages.html

《Xposed中的XSharePreference的使用》

《Xposed 使用注意事项》

《Xposed Xposed读取SharedPreferences》