# 鸿蒙开源第三方组件——VideoCache视频缓存组件

## 前言

基于安卓平台的视频缓存组件VideoCache（ https://github.com/danikula/AndroidVideoCache），实现了鸿蒙化迁移和重构，代码已经开源到（https://gitee.com/isrc\_ohos/android-video-cache\_ohos），欢迎各位下载使用并提出宝贵意见！

## 背景

用户在网速波动较大的环境下浏览视频时，经常会遇到由于网速较慢引起的持续加载或播放失败的情况。VideoCache组件实现了视频缓存功能，播放视频的同时，对视频源进行缓存。出现网速较慢的情况时，手机读取提前缓存好的视频数据，可以保证视频的正常播放，给予用户更流畅的观看体验。

## 组件效果图展示

### 1、主菜单界面: 视频播放

安装软件后，只需要在鸿蒙设备上单击HarmonyVideoCache软件图标，打开软件即可进入主菜单界面，进入主菜单界面后会自动开始播放视频，如下图所示：

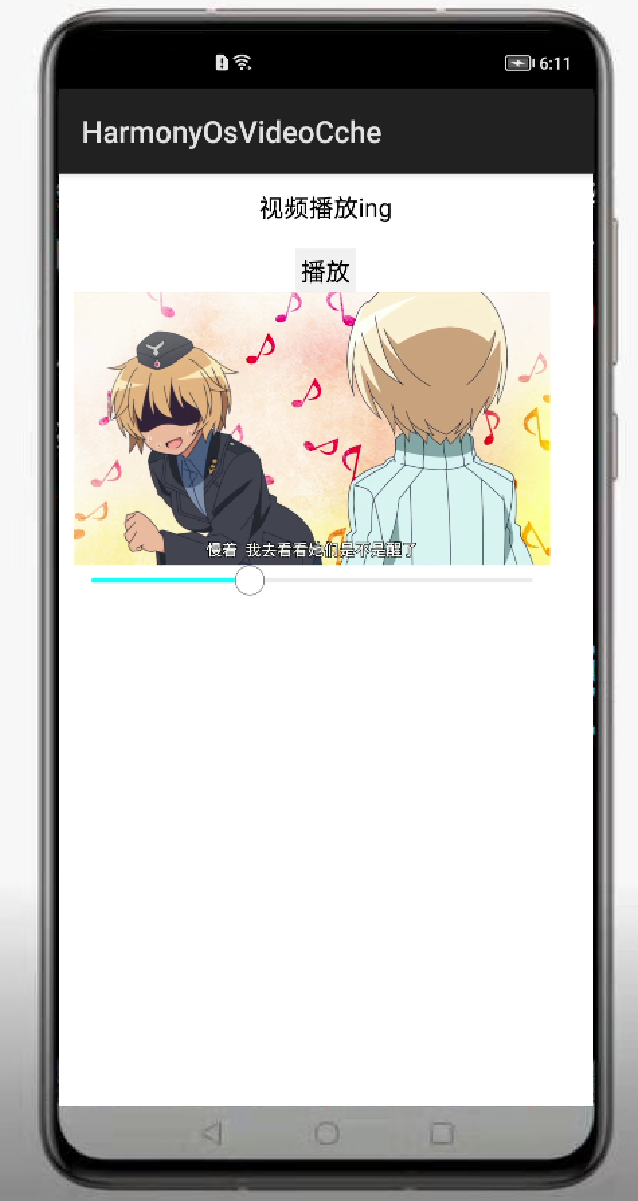
****

图 1 视频播放的主菜单界面

### 2、验证缓存

等待视频播放完成后，可以手动关闭手机的数据连接和WIFI连接：



图 2 关闭网络连接

在关闭了网络连接之后，回到VideoCache应用中，点击播放按钮, 会发现视频是可以通过本地缓存重新播放的。注意到图1和图3的区别，在图1中任务栏可以看到有WIFI连接显示，图3 中没有WIFI连接。

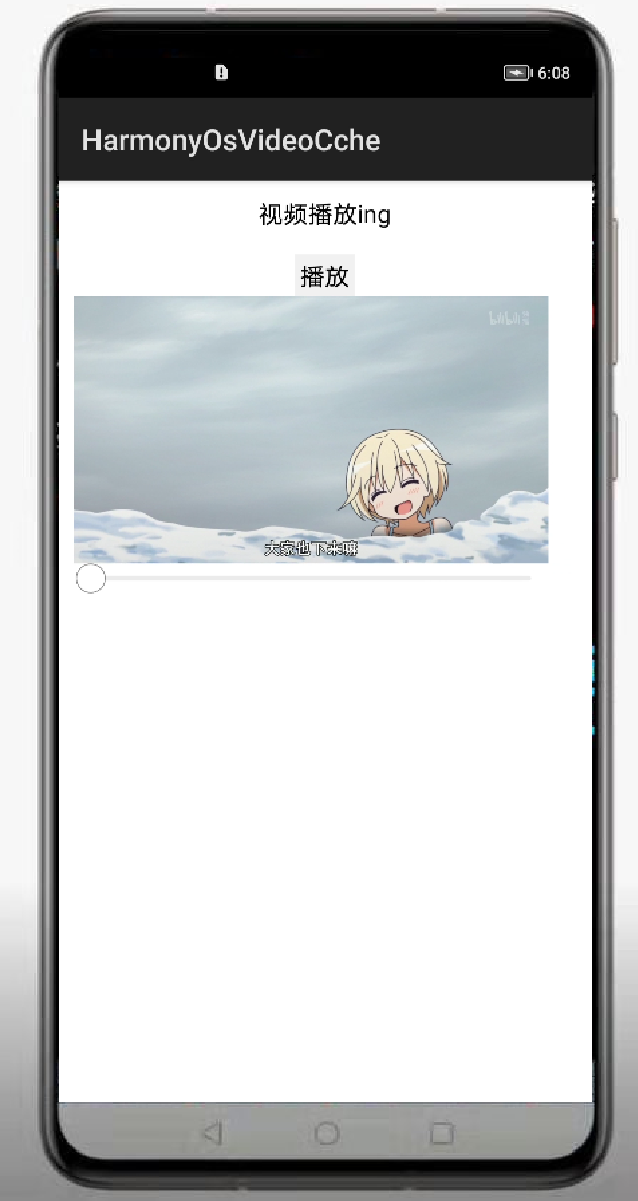


图 3 缓存播放视频

## Sample解析

如图4所示，该组件在本地与远程服务器之间建立了代理服务器。当本地发送视频网络请求至代理服务器时，代理服务器与远程服务器之间通过代理Socket连接，并将远程服务器的视频数据回写到代理服务器的缓存中，本地播放视频时从代理服务器的缓存中读取数据（图4援引自https://www.jianshu.com/p/4745de02dcdc）。下面详细介绍视频缓存的步骤。

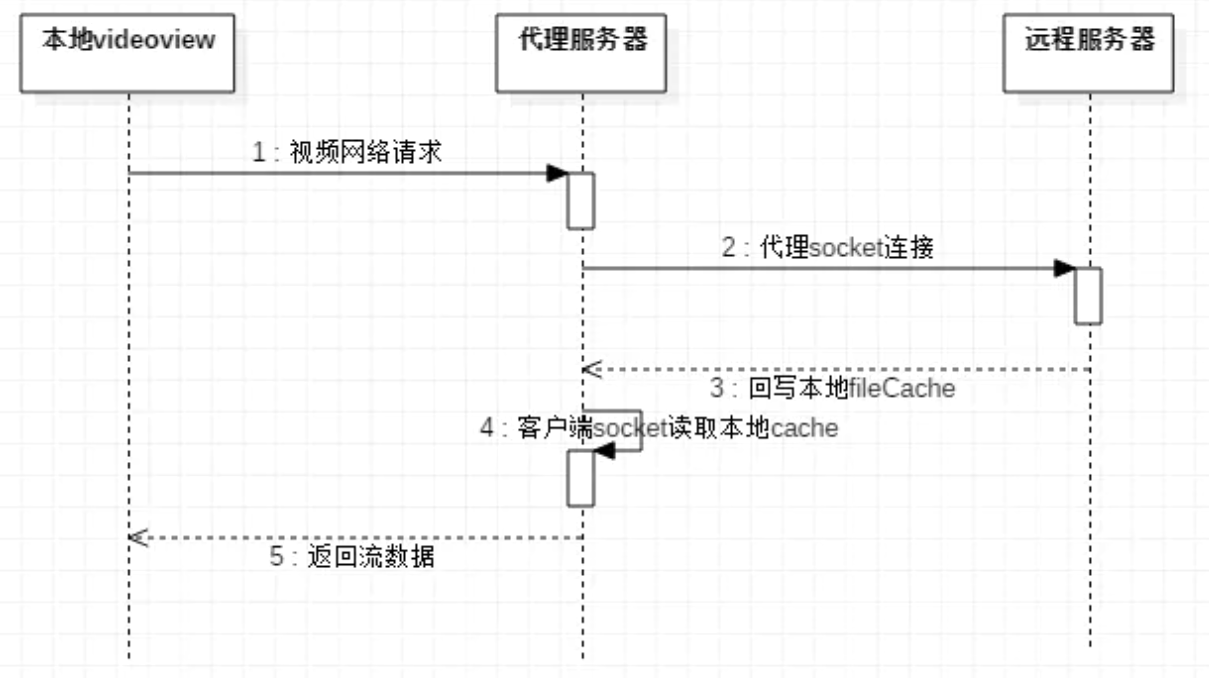


图4 VideoCache组件的视频缓存原理

1、实例化HttpProxyCacheServer类的对象

HttpProxyCacheServer类可用于处理来自视频播放器的播放请求，当本地有缓存时，向视频播放器返回一个本地IP地址（LocalURL：以127.0.0.1开头），用于视频的播放。

**private** HttpProxyCacheServer mCacheServerProxy=**null**;

**public void** onStart(Intent intent) {  
 ...**if** (mCacheServerProxy == **null**) {  
 Context context = **this**;

//实例化HttpProxyCacheServer对象  
 mCacheServerProxy = **new** HttpProxyCacheServer(context);  
 }

...

}

2、定义缓存监听器CacheListener

CacheListener 用于监听文件缓存的进度，方便开发者通过判断缓存进度，执行各类操作。

onCacheAvailable()方法是设置CacheListener 监听器时需要重写的方法，此方法的参数中：cacheFile表示缓存文件的地址；url表示网络视频的URL；percentsAvailable表示缓存进度，取值为1~100，取值为100时表示全部视频缓存完成。

基于percentAvailable变量，大多数视频播放器有以下设计：设置一个变量用于保存当前的视频播放进度。在缓存监听器CacheListener 中，比较当前缓存进度与当前播放进度的差值，如果超出了预设值，可以执行特定操作以暂停缓存，直至二者的差值小于预设值，重新启动缓存。

//缓存相关  
**private** CacheListener mCacheListener = **new** CacheListener() {  
 @Override  
 **public void** onCacheAvailable(File cacheFile, String url, **int** percentsAvailable) {

//打印实时缓存进度  
 HiLog.info(**new** HiLogLabel(3,0,**"cache"**),**"Saving……,percent:"**+String.valueOf(percentsAvailable));

//当进度达到100时，可进行一些特殊操作，此处仅以log打印为例  
 **if** (percentsAvailable == 100 && !cacheFile.getPath().endsWith(**".download"**)) {  
 HiLog.info(**new** HiLogLabel(3,0,**"cache"**),**"Download already!"**);  
 }  
 }  
};

3. 获取LocalURL

将网络视频的URL与步骤2中的监听器对象mCacheListener传入HttpProxyCacheServer类的注册方法中，即可对缓存进行监听。后通过 HttpProxyCacheServer类的getProxyUrl()方法获取网络视频URL对应的LocalUrl。

//注册下载缓存监听

mCacheServerProxy.registerCacheListener(mCacheListener,URL);

//获取LocalURL

localUrl = mCacheServerProxy.getProxyUrl(URL);

1. 使用LocalUrl作为视频来源进行播放，缓存功能即可实现。

## Library解析

整个library分为五个部分：file、headers、slice、sourcestorage以及22个类文件，如图2所示。

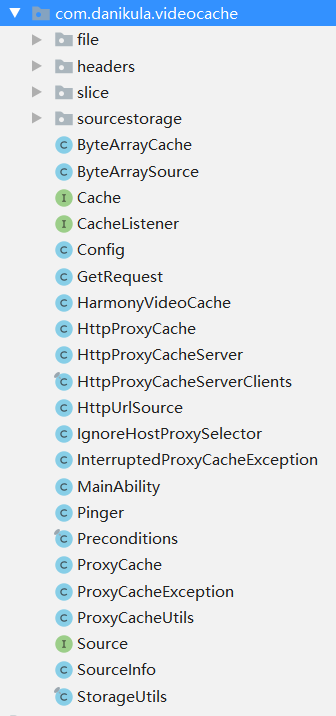


图5 library的组成结构

### 一、file

在file文件夹下的类主要涉及文件缓存相关的功能：

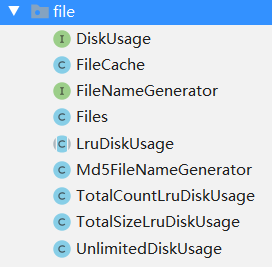


图6 file文件夹的组成结构

1. FileCache类

类中规定了缓存文件的命名格式（后加.download）和存储的路径，完成了缓存文件的创建。

//定义缓存文件的后缀格式

**private static final** String TEMP\_POSTFIX = **".download"**;  
**public** FileCache(File file, DiskUsage diskUsage) **throws** ProxyCacheException {  
 ...

File directory = file.getParentFile();  
 Files.makeDir(directory);  
 **boolean** completed = file.exists();

//文件的保存格式：根目录文件+文件名+之前定义的文件后缀格式  
 **this**.file = completed ? file : **new** File(file.getParentFile(), file.getName() + TEMP\_POSTFIX);

//文件权限设置。缓存完成，文件只能读取；未缓存完成，文件可读可写。  
 **this**.dataFile = **new** RandomAccessFile(**this**.file, completed ? **"r"** : **"rw"**);  
} **catch** (IOException e) {  
 **throw new** ProxyCacheException(**"Error using file "** + file + **" as disc cache"**, e);  
}

1. Files类

此类是对JAVA中原有的File类的封装，原File类仅可处理一个文件，Files类可同时对多个文件进行处理。

如下代码中，getLruListFiles()方法的参数是一个directory，在方法中对directory（文件夹路径）下的所有文件进行拆分，返回了一个File参数类型的List列表，后续可对列表中的各个File文件进行处理。

**static** List<File> getLruListFiles(File directory) {

//通过list对Files内的文件进行处理  
 List<File> result = **new** LinkedList<>();  
 File[] files = directory.listFiles();

//为各file建立LastModifiedComparator

//LastModifiedComparator可用于根据文件的上次修改的日期文件进行排序  
 **if** (files != **null**) {  
 result = Arrays.asList(files);  
 Collections.sort(result, **new** LastModifiedComparator());  
 }  
 **return** result;  
}

1. LruDiskUsage类

此类主要用于控制缓存文件的大小，它与Videocache平行开了一个线程，实时记录缓存文件的数量、大小、存储空间等，超过预设的阈值时，执行特定的优化操作。

**private void** trim(List<File> files) {

**long** totalSize = countTotalSize(files); //缓存文件的总大小  
 **int** totalCount = files.size(); //缓存文件的总数量  
 **for** (File file : files) {

//未超过缓存文件的（总大小 & 总数量）的阈值时，接收缓存  
 **boolean** accepted = accept(file, totalSize, totalCount);  
 **if** (!accepted) {

**long** fileSize = file.length(); // 单一文件的大小  
 **boolean** deleted = file.delete(); //文件是否为**预备删除**的文件

//如果是准备删除的文件

**if** (deleted) {  
 totalCount--; // 缓存文件的总数量-1  
 totalSize -= fileSize; //缓存文件的总大小 - **预备删除**的单一文件的大小

LOG.info(**"Cache file "** + file +

**" is deleted because it exceeds cache limit"**);  
 } **else** {  
 LOG.error(**"Error deleting file "** + file + **" for trimming cache"**);  
 }  
 }  
 }  
}

1. Md5FileNameGenerator类

此类实现了为输入文件路径，生成对应的MD5值的功能。MD5值是一种被"压缩"的保密格式，可以确保信息完整传输。

**public class** Md5FileNameGenerator **implements** FileNameGenerator {  
 **private static final int** MAX\_EXTENSION\_LENGTH = 4;  
 @Override  
 **public** String generate(String url) {

//获取文件名的后缀  
 String extension = getExtension(url);

//获取MD5值

String name = ProxyCacheUtils.computeMD5(url);  
 Boolean isEmpty = **false**;

//文件后缀名为空时，设置isEmpty 标志位为true  
 **if** (extension == **null** || extension.length() == 0)   
 isEmpty = **true**;**return** isEmpty ? name : name + **"."** + extension;  
 }

1. TotalCountLruDiskUsage类、TotalSizeLruDiskUsage类和UnlimitedDiskUsage类

LruDiskUsage类是标题中前两个类的父类，同时控制缓存文件的大小和数量，需要判断当前缓存文件的（总大小 & 总数量）未超过阈值时，才会缓存新的文件。Total**Count**LruDiskUsage类和Total**Size**LruDiskUsage类分别只对缓存文件总数量或者缓存文件总大小进行限制，满足一个条件便可以缓存新的文件。

TotalCountLruDiskUsage类和TotalSizeLruDiskUsage类各有两个方法：一个方法用于设定缓存文件的阈值；一个方法用于判断当前缓存数据是否超过了设定的阈值。

当不需要进行磁盘的缓存限制时使用UnlimitedDiskUsage类，其本身是一个空的类，不对缓存文件的数量和大小做任何限制。

//控制缓存文件的总数量

**public class** TotalCountLruDiskUsage **extends** LruDiskUsage {  
 **private final int** maxCount;  
 //设置缓存文件的总数量的阈值  
 **public** TotalCountLruDiskUsage(**int** maxCount) {  
 **if** (maxCount <= 0) {  
 **throw new** IllegalArgumentException(**"Max count must be positive number!"**);  
 }  
 **this**.maxCount = maxCount;  
 }

//当前缓存文件的总数量小于设定的阈值时，新文件accept  
 @Override  
 **protected boolean** accept(File file, **long** totalSize, **int** totalCount) {  
 **return** totalCount <= maxCount;  
 }  
}

//控制制缓存文件的总大小

**public class** TotalSizeLruDiskUsage **extends** LruDiskUsage {  
 **private final long** maxSize;  
 //设置制缓存文件的总大小的阈值  
 **public** TotalSizeLruDiskUsage(**long** maxSize) {  
 **if** (maxSize <= 0) {  
 **throw new** IllegalArgumentException(**"Max size must be positive number!"**);  
 }  
 **this**.maxSize = maxSize;  
 }

//当前缓存文件的总大小小于设定的阈值时，新文件accept  
 @Override  
 **protected boolean** accept(File file, **long** totalSize, **int** totalCount) {  
 **return** totalSize <= maxSize;  
 }  
}

### 二、headers

文件中涉及到的功能不多，仅有一个接口文件和一个能实现URL和文件路径hashmap匹配功能的类文件，上述功能在HttpProxyCacheServer类中被调用。

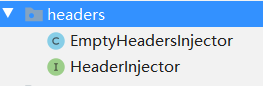


图7 headers文件夹的组成结构

### slice

鸿蒙程序的slice控件用于三方件迁移中的可视化调试，在这里我们对其不作进一步的分析。



图8 slice文件夹的组成结构

### sourcestorage

sourcestorage用于在数据库中存储SourInfo。SourInfo可用于存储http请求源的一些信息，如URL，数据长度Length，请求资源的类型MIME等。sourcestorage中的类主要在上述的HttpProxyCacheServer类中被调用。

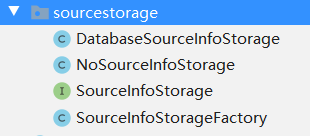


图9 sourcestorage文件夹的组成结构

DatabaseSourceInfoStorage类用于做数据库的初始化工作，数据库里面存的字段主要是URL、Length、MIME，SourceInfo类是对这3个字段的封装。类中包含了三个接口：get()、 put()、release()，可供外部调用，三个接口都是对SourceInfo的操作，主要用来查找和保存缓存的信息。

其余三个类是根据DatabaseSourceInfoStorage类进行的工厂模式的生成，如果对这部分不明白的同学可以在网上搜索“设计模式-工厂模式”进行学习。

**class** DatabaseSourceInfoStorage **extends** DatabaseHelper

**implements** SourceInfoStorage {  
 //数据库中存储SourInfo：URL、Length、MIME  
 **private static final** String TABLE = **"SourceInfo"**;  
 **private static final** String COLUMN\_ID = **"\_id"**;  
 **private static final** String COLUMN\_URL = **"url"**;  
 **private static final** String COLUMN\_LENGTH = **"length"**;  
 **private static finavl** String COLUMN\_MIME = **"mime"**;  
 **private static final** String[] ALL\_COLUMNS = **new** String[]{COLUMN\_ID, COLUMN\_URL,

COLUMN\_LENGTH, COLUMN\_MIME};

//创建数据库的SQL  
 **private static final** String CREATE\_SQL =  
 **"CREATE TABLE "** + TABLE + **" ("** +  
 COLUMN\_ID + **" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,"** +  
 COLUMN\_URL + **" TEXT NOT NULL,"** +  
 COLUMN\_MIME + **" TEXT,"** +  
 COLUMN\_LENGTH + **" INTEGER"** +  
 **");"**;  
  
 **private final** RdbStore myRdbStore;

//连接的数据库名字  
 **private final** StoreConfig config =

StoreConfig.newDefaultConfig(**"AndroidVideoCache.db"**);

}

//数据库get指令，通过URL获取SourceInfo

**public** SourceInfo get(String url) {  
 checkNotNull(url);  
 ResultSet cursor = **null**;  
 **try**{  
 RdbPredicates predicates = **new** RdbPredicates(TABLE);  
 predicates.equalTo(COLUMN\_URL, url);  
 cursor = **this**.myRdbStore.query(predicates, **null**);  
 **return** cursor == **null** || !cursor.goToFirstRow() ? **null** : convert(cursor);  
 } **finally** {  
 **if** (cursor != **null**) {  
 cursor.close();  
 }  
 }  
}

//数据库put指令，将url和SourceInfo在数据库中登记绑定

**public void** put(String url, SourceInfo sourceInfo) {  
 checkAllNotNull(url, sourceInfo);  
 SourceInfo sourceInfoFromDb = get(url);  
 **boolean** exist = sourceInfoFromDb != **null**;  
 RdbPredicates predicates = **new** RdbPredicates(TABLE);  
 **if** (exist) {  
 predicates.contains(COLUMN\_URL, url);  
 **this**.myRdbStore.update(convert(sourceInfo), predicates);  
 } **else** {  
 **this**.myRdbStore.insert(TABLE, convert(sourceInfo));  
 }  
}  
//release指令：释放数据库控制流  
@Override  
**public void** release() {  
 **this**.myRdbStore.close();  
}

### 主功能文件

这部分文件主要用于整合上述四个部分的功能，向外部提供VideoCache接口。

主要功能类如下图所示，他们的外部调用方法在Sample中已经详细说明，主要使用到的就是HttpProxyCacheServer类，下面对其内部实现进行详细的讲解。

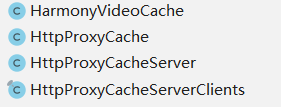


图10 主要功能类主文件

1、构造函数

在构造函数中主要进行了全局变量的初始化和对PROXY\_HOST（VideoCache代理接口，也就是LocalURL所属的代理接口）进行访问，判断是否可以直接ping通。

**private** HttpProxyCacheServer(Config config) {  
 **this**.config = checkNotNull(config);  
 **try** {

//初始化各种全局变量  
 InetAddress inetAddress = InetAddress.getByName(PROXY\_HOST);  
 **this**.serverSocket = **new** ServerSocket(0, 8, inetAddress);  
 **this**.port = serverSocket.getLocalPort();  
 IgnoreHostProxySelector.install(PROXY\_HOST, port);  
 CountDownLatch startSignal = **new** CountDownLatch(1);  
 **this**.waitConnectionThread = **new** Thread(**new** WaitRequestsRunnable(startSignal));  
 **this**.waitConnectionThread.start();  
 startSignal.await(); *// freeze thread, wait for server starts*

//获取对PROXY\_HOST& port的ping，判断是否可以ping通  
**this**.pinger = **new** Pinger(PROXY\_HOST, port);  
 LOG.info(**"Proxy cache server started. Is it alive? "** + isAlive());  
 } **catch** (IOException | InterruptedException e) {  
 socketProcessor.shutdown();  
 **throw new** IllegalStateException(**"Error starting local proxy server"**, e);  
 }  
}

2、registerCacheListener函数

这个函数主要实现的功能是对URL进行注册监听。

**public void** registerCacheListener(CacheListener cacheListener, String url) {  
 checkAllNotNull(cacheListener, url);  
 **synchronized** (clientsLock) {  
 **try** {

//对url获取Clients，并为其注册CacheListener  
 getClients(url).registerCacheListener(cacheListener);  
 } **catch** (ProxyCacheException e) {  
 LOG.warn(**"Error registering cache listener"**, e);  
 }  
 }  
}

3、getProxyUrl函数

该函数实现了将（已经注册过的）URL转化为cached LocalURL的功能。

**public** String getProxyUrl(String url) {  
 **return** getProxyUrl(url, **true**);  
}

**public** String getProxyUrl(String url, **boolean** allowCachedFileUri) {  
 **if** (allowCachedFileUri && isCached(url)) {  
 File cacheFile = getCacheFile(url);  
 touchFileSafely(cacheFile);  
 **return** Uri.getUriFromFile(cacheFile).toString();  
 }  
 **return** isAlive() ? appendToProxyUrl(url) : url;  
}

当传入一个网络视频的URL时，该方法会对该URL进行判断，如果可以在代理服务器上进行缓存，则提供正确的LocalURL返回值，否则返回原URL。

## 项目贡献人

吕泽 郑森文 朱伟 陈美汝 张馨心