新浪微盘 VDisk
Android SDK 开发者文档

时间	内容	参与人
2013-5-7	文档创建，初版完成	殷凯，王彤
2014-2-11	添加微博 SSO 认证登录	王彤
2014-2-25	添加使用 RefreshToken 刷新 AccessToken 机制	王彤

微盘 Android SDK 开发者交流群： 240235926
微盘 OpenAPI、全平台 SDK 交流群： 134719337, 162285095

请先前往微盘开发者中心注册为微盘开发者，并创建应用。

认证相关

OAuth2.0（开放授权）是一个开放标准，用户授权后，第三方应用无需获取用户的用户名和密码就可以访问该用户在某一网站上存储的资源（如照片，视频和其他文件）。

以下介绍操作步骤：

1. 在 Eclipse 中导入 VDiskSdk 项目。

2. 在您的项目（如果已经存在）右键 Properties - Android - Library - Add 添加 VDiskSdk。

3. 在您的项目的 AndroidManifest.xml 中加入以下可能需要使用的权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

4. 在您的项目里负责认证的 Activity 中继承接口 implements VDiskDialogListener。

5. 在您的项目中填写开发者应用的 App Key 及 App Secret，开发者可登录新浪微盘开放平台 http://vdisk.weibo.com/developers/进入“我的应用”查看。

例如

```
public static final String CONSUMER_KEY = "//填入开发者应用的App Key";
public static final String CONSUMER_SECRET = "//填入开发者应用的 App Secret";
```

设定与 App Key 对应的应用回调地址，此地址可在开发者登陆新浪微盘开发平台后，进入“我的应用 - 编辑应用信息 - 回调地址”进行设置和查看，如需使用微盘 Token 方式认证登陆，则该应用回调页不可为空。

```
private static final String REDIRECT_URL = "//填入开发者应用的回调地址";
```

已经使用或需要配合使用新浪微博 SDK 的应用开发者，如果已经获取用户的微博 Token，也可以选择微博 Token 方式进行微盘的认证和登录。微博 Token 是一个以”2.00”为开头的字符串。如需使用微博 Token 直接访问微盘的 API，则这个字段不能为空。

```
public static final String WEIBO_ACCESS_TOKEN = "2.00xbOE8IOZ_.......";
```

在使用新浪微博 Token 直接登录的时候，SDK 本身并不直接在认证阶段检查 Token 字符串，开发者需保证获取到的微博 Token 的正确性，否则可能会接收到如下异常信息：

VDiskServerException (nginx/1.0.5): 504 Gateway Timeout (50403: Dependent service failed.)

6.全局声明一个会话,并在您的项目 onCreate()时对其进行初始化：

```
VDiskAuthSession session;
AppKeyPair appKeyPair = new AppKeyPair(CONSUMER_KEY, CONSUMER_SECRET);
session = VDiskAuthSession.getInstance(this,appKeyPair,AccessType.APP_FOLDER);
```

对session进行创建时，注意到第三个参数AccessType.APP_FOLDER，代表适合普通开发者使用的沙箱（SandBox）模式，应用在此模式下对用户微盘空间的改动与微盘官方Web站上的内容完全隔离，仅共享存储空间。用户空间隔离详情参见权限规则与说明：

http://vdisk.weibo.com/developers/index.php?module=api&action=rights

另一种参数为AccessType.VDISK的模式为basic访问模式，应用在此模式下可读写用户微盘空间的公有数据，仅提供给部分拥有高级权限AppKey的开发者，如需为应用申请basic模式权限，详情请访问：

http://vdisk.weibo.com/developers/index.php?module=developers&action=space

7.直接使用微盘登录的，需设置回调地址，并执行认证方法：

```
session.setRedirectUrl(REDIRECT_URL);
session.authorize(OAuthActivity.this, OAuthActivity.this);
```

更新：使用新浪微博 Token 认证：

2014 年 2 月微盘 SDK 版本更新后，集成了新浪微博 Android SDK 的 SSO 认证登录功能：http://open.weibo.com/wiki/SDK#Android_SDK，开发者可通过调用用户手机上的微博客户端直接获取微博 Token 进行微盘认证，免去用户输入帐号密码的过程，也降低微盘开发者的开发负担。

如果您目前已经是集成了微博 SDK 的开发者，您可以继续按照原有流程进行开发，即在获取微博 Access token 后，执行以下认证方法：

```
WeiboAccessToken weiboToken = new WeiboAccessToken();
weiboToken.mAccessToken = OAuthActivity.WEIBO_ACCESS_TOKEN;
session.enabledAndSetWeiboAccessToken(weiboToken);
session.authorize(OAuthActivity.this, OAuthActivity.this);
```

如果您目前不是微博 SDK 的开发者，又希望快捷地开发使用微博 Token 登录微盘的功能，可以执行如下流程：

（1）在您项目的/libs文件夹下加入weibosdkcore.jar文件（位于VDiskSdk_example/libs），此为微博Open API官方SDK的闭源接口包；

（2）添加VDiskSdk_Example项目中的com.sina.weibo.sdk.demo包及其下的类文件，此包为从Weibo Open API官方SDK中抽取与认证登录相关的类；

（3）前往新浪微博开放平台http://open.weibo.com/注册成为微博开发者，进入管理中心 - 我的应用，并创建应用，在com.sina.weibo.sdk.demo.Constants类中填写您微博应用的APP_KEY和回调地址REDIRECT_URL。

（4）在新浪微博开放平台http://open.weibo.com/进入管理中心 - 我的应用 - 应用控制台-应用信息-应用基本信息编辑，填写Android包名、签名及下载地址，其中签名项可使用微博官方SDK提供的签名工具获取：https://github.com/mobileresearch/weibo_android_sdk项目中的app_signatures.apk。

（5）代码中执行

```
WeiboAuth mWeiboAuth;
mWeiboAuth = new WeiboAuth(this,Constants.APP_KEY,
    Constants.REDIRECT_URL, Constants.SCOPE);

mSsoHandler = new SsoHandler(OAuthActivity.this, mWeiboAuth);
mSsoHandler.authorize(new AuthListener());
```

回调类AuthListener继承了WeiboAuthListener，在其回调方法onComplete()中可获得认证完成后的微博Access Token，之后即可按照以下流程进行微盘认证：

```
WeiboAccessToken weiboToken = new WeiboAccessToken();
weiboToken.mAccessToken = OAuthActivity.WEIBO_ACCESS_TOKEN;
session.enabledAndSetWeiboAccessToken(weiboToken);
session.authorize(OAuthActivity.this, OAuthActivity.this);
```

具体实现可参考VDiskSdk_example中com.vdisk.android.example.OAuthActivity.java

（6）另外，请注意您微博应用的OAuth2.0授权有效期，测试级别的授权有效期仅保证您的用户微博Token有效1天，具体可在微博开放平台 - 接口管理 - 授权机制查看。

8.判断连接成功后，即可登入并执行您的其他操作

```
if (session.isLinked()) {
            startActivity(new Intent(this, VDiskTestActivity.class));
            finish();
        }
```

9.更新：使用RefreshToken刷新微盘AccessToken机制：

若在初始化阶段，当前用户的微盘AccessToken已过期，不需要用户再次进行认证，可通过当前的RefreshToken获取新的微盘AccessToken：

```
AccessToken mVDiskAccessToken;
mVDiskAccessToken = VDiskAuthSession.getOAuth2Preference(this,appKeyPair);
mVDiskAccessToken = session.refreshOAuth2AccessToken(
                    mVDiskAccessToken.mRefreshToken,OAuthActivity.this);
```

具体实现可参考 VDiskSdk_example 中 com.vdisk.android.example.OAuthActivity.java 中 refreshLogin()方法。

刷新成功服务器将返回新的微盘 AccessToken，其有效期及新的 RefreshToken，可调用 session.updateOAuth2Preference(OAuthActivity.this,mVDiskAccessToken); 将新的信息存储，便于下一次登录或刷新。具体实现可参考 DiskSdk_example 中 com.vdisk.android.example.OAuthActivity.java 中 handler 对象。

10.以下方法需要重载

```
/**
     * 认证结束后的回调方法
     *
     * Callback method after authentication.
     */
    @Override
    public void onComplete(Bundle values) {

        if (values != null) {
            AccessToken mToken = (AccessToken) values
                    .getSerializable(VDiskAuthSession.OAUTH2_TOKEN);
            session.finishAuthorize(mToken);
        }
        startActivity(new Intent(this, VDiskTestActivity.class));
        finish();
    }

    /**
     * 认证出错的回调方法
     *
     * Callback method for authentication errors.
     */
    @Override
    public void onError(VDiskDialogError error) {
        Toast.makeText(getApplicationContext(),
                "Auth error : " + error.getMessage(), Toast.LENGTH_LONG).show();
    }

    /**
     * 认证异常的回调方法
     *
     * Callback method for authentication exceptions.
     */
    @Override
    public void onVDiskException(VDiskException exception) {
```

```
        Toast.makeText(getApplicationContext(),
                "Auth exception : " + exception.getMessage(), Toast.LENGTH_LONG)
                .show();
    }

    /**
     * 认证被取消的回调方法
     *
     * Callback method as authentication is canceled.
     */
    @Override
    public void onCancel() {
        Toast.makeText(getApplicationContext(), "Auth cancel",
                Toast.LENGTH_LONG).show();
    }
```

用户信息相关接口

1.在使用新浪微盘SDK所有相关API前，务必声明并初始化VDiskAPI。
声明：

```
VDiskAPI<VDiskAuthSession> mApi;
```

与认证部分类似，需使用开发者应用的AppKey和AppSecret对全局会话session进行初始化：

```
VDiskAuthSession session;

AppKeyPair appKeyPair = new AppKeyPair(OAuthActivity.CONSUMER_KEY,
                OAuthActivity.CONSUMER_SECRET);

session = VDiskAuthSession.getInstance(this, appKeyPair,
                AccessType.APP_FOLDER);
```

并使用session初始化mApi：

```
mApi = new VDiskAPI<VDiskAuthSession>(session);
```

2.获取用户微盘空间信息，用于开发者协调自身程序所需的存储方式：

```
Account account = mApi.accountInfo();
```

account.quota （long）用户微盘总空间
Account.consumed （long）用户微盘已使用空间

3.用户会话注销：

```
session.unlink();
```

此操作将清除本地保存的微盘或微博 Token 信息

上传下载模块

1.小文件上传

使用

VDiskAPI.putFileOverwriteRequest(String path, InputStream is, long length, ProgressListener listener) throws VDiskException

发起一次小文件上传请求，并可手动中断，如遇重复文件则将其覆盖。

Parameters:

path the full VDisk path where to put the file, including directories and filename.

is the InputStream from which to upload.

length the amount of bytes to read from the InputStream.

listener an optional ProgressListener to receive upload progress updates, or null.

Returns:

an UploadRequest.

其是对

putFileRequest(String path, InputStream is, long length, boolean overwrite, String parentRev, ProgressListener listener) throws VDiskException

的封装。如果不希望覆盖，可修改参数值 **boolean overwrite** 为 false。

例如

```
UploadRequest mRequest;
mRequest = mApi.putFileOverwriteRequest(path, fis, mFile.length(),
                    new ProgressListener() {
                        @Override
                        public long progressInterval() {
                            // 在这里设置进度更新间隔，缺省为0.5秒
                            return super.progressInterval();
                        }

                        @Override
                        public void onProgress(long bytes, long total) {
                            // 在这里可以处理进度更新
                            // 参数bytes：当前已完成上传的字节数
                            // 参数total：总字节数
                            // 例如 publishProgress(bytes);
                        }
                    });
```

另外
调用 mRequest.abort();可将上传会话中断，并将抛出异常

```
catch (VDiskPartialFileException e) {
// 开发者可捕获该异常并做相应处理
}
```

具体实现可参考 VDiskSdk_example 中 com.vdisk.android.example.UploadPicture.java

2.大文件分段上传

大文件分段上传由 Sina Simple Storage Service（Sina S3）服务器及微盘服务器的接口支持，在开发 Android 应用时可调用 SDK 的相应 API 实现。

使用

```
VDiskAPI.putLargeFileOverwriteRequest(String srcPath, String desPath, long length, ComplexUploadHandler handler) throws VDiskException
```

将较大的文件分段后上传。

```
Parameters:
    srcPath the local path of the file.
    desPath the full VDisk path where to put the file, including directories and filename.
    length the length of the local file.
    handler the handler to control the upload session.
```

其是对

```
VDiskAPI.putLargeFileRequest(String srcPath, String desPath, long length, long segmentLength, boolean overwrite, String parentRev, ComplexUploadHandler handler) throws VDiskException
```

的封装。如果不希望覆盖，可修改参数值 boolean overwrite 为 false。

例如

```
putLargeFileRequest(srcPath, desPath, length, -1, true, null, handler);
```

其中参数 long segmentLength 为文件分段长度，如小于 1，则采用默认分片大小 4MB
long com.vdisk.net.VDiskAPI.UPLOAD_DEFAULT_SECTION_SIZE = 4194304 [0x400000]

其中参数 ComplexUploadHandler handler 是一个用于实时获取上传进度和状态的控制协调类。同时，通过此类，可以读取，更新和删除本地数据库中存储的上传分段信息。

例如

```
handler = new ComplexUploadHandler(mContext) {
                @Override
                public void onProgress(long bytes, long total) {
                    // 在这里可以处理进度更新
```

```
                    // 参数bytes：当前已完成上传的字节数
                    // 参数total：总字节数
                    // 例如 publishProgress(bytes);
                }
```

使用

```
@Override
                public void startedWithStatus(ComplexUploadStatus status) {
                    switch (status) {
                    case ComplexUploadStatusLocateHost:
                        Log.d(TAG, "Getting the nearest host...");
                        break;
                    case ComplexUploadStatusCreateFileSHA1:
                        Log.d(TAG, "Creating the sha1 of file");
                        break;
                    case ComplexUploadStatusInitialize:
                        Log.d(TAG, "Signing each segment of file...");
                        break;
                    case ComplexUploadStatusCreateFileMD5s:
                        Log.d(TAG, "Creating each segment's md5...");
                        break;
                    case ComplexUploadStatusUploading:
                        Log.d(TAG, "Uploading one segment...");
                        break;
                    case ComplexUploadStatusMerging:
                        Log.d(TAG, "File Merging...");
                        break;
                    default:
                        break;
                    }
```

开发者可获取当前大文件的上传状态，并做相应操作。

大文件分段上传流程：

（1）首先通过 http://up.sinastorage.com/?extra&op=domain.json 获取离当地最近的 Sina Simple Storage Service（Sina S3）服务器的域名。

（2）计算要上传文件的 sha1 值，用于与服务器的文件列表比对，如果有一致情况，则会调用秒传接口，由服务器端直接做文件的逻辑搬移工作，秒传成功可直接返回。

（3）获取批量签名信息，以及此文件的每一个分段将要上传到的分布服务器地址。文件被分为若干段上传时，每一段有均可能传向不同的地址，并在完成上传时由 Sina S3 服务器完成文件分段的合并。

（4）创建每一个文件分段的 md5 值，用于同服务器验证每一个上传文件的分段是否保持正确性和完整性。

（5）开始上传文件分段。每个分段成功时将会将当前进度信息写入数据库。

（6）完成文件所有分段的上传，访问服务器端合并文件的接口。并将本文件的数据库信息清除。

若当此文件上传人为或意外中断，再次上传时将检查数据库中是否存在该文件的上传会话信息，如果有上传记录且未过期，则从上次成功的文件分片计数之后开始，进行续传。

数据库中仅存储了两个字段值，file_id 和 file_obj。
file_id，MD5(上传文件本地路径 +上传文件的微盘目标路径)，用来标识待上传的文件;
file_obj，表示文件对象序列化后的形式。

开发者如需添加处理其他字段可修改数据表，并可通过重载以下方法自定义本地文件分段信息的存储方式，及对应操作：

读取数据库信息：

```
VDiskUploadFileInfo
com.vdisk.android.ComplexUploadHandler.readUploadFileInfo(String srcPath,
String desPath) throws VDiskException
```

更新数据库信息：

```
void
com.vdisk.android.ComplexUploadHandler.updateUploadFileInfo(VDiskUploadFileInf
o fileInfo) throws VDiskException
```

删除数据库信息：

```
void
com.vdisk.android.ComplexUploadHandler.deleteUploadFileInfo(VDiskUploadFileInf
o fileInfo)
```

另外
调用 handler.abort();可将上传会话中断，并将抛出异常

```
catch (VDiskPartialFileException e) {
// 开发者可捕获该异常并做相应处理
}
```

具体流程可参考 VDiskSdk_example 中 com.vdisk.android.example.LargeFileUpload.java

3.文件下载
使用

```
VDiskAPI.createDownloadDirFile(String targetPath)
```

创建一个下载文件的存储位置。

使用

VDiskFileInfo com.vdisk.net.VDiskAPI.getFile(String path, String rev, OutputStream os, File targetFile, ProgressListener listener) throws VDiskException

将文件从微盘服务器中下载下来并写入本地文件中。

Parameters:

path the VDisk path to the file.

rev the revision (from the file's metadata) of the file to download, or null to get the latest version.

os the OutputStream to write the file to.

listener an optional ProgressListener to receive progress updates as the file downloads, or null.

targetFile

Returns:

the VDiskFileInfo for the downloaded file.

例如

首先在本地创建一个下载文件存放位置，并建立一个文件输出流。

```
file = mApi.createDownloadDirFile(mTargetPath);
try {
    mFos = new FileOutputStream(file, true);
    } catch (FileNotFoundException e) {
    mErrorMsg = "Couldn't create a local file to store the file";
    return false;
}
```

其中 createDownloadDirFile 方法将在本地目录创建一个名为“目标文件名.vdisktemp”的临时文件。

当文件下载成功后，将临时后缀.vdisktemp 去掉，即恢复目标文件名。

```
private VDiskFileInfo info;
info = mApi.getFile(mPath, null, mFos, file,
                    new ProgressListener() {

                        @Override
                        public long progressInterval() {
                            // 在这里设置进度更新间隔，缺省为0.5秒
                            return super.progressInterval();
                        }

                        @Override
                        public void onProgress(long bytes, long total) {
                        // 在这里可以处理进度更新
                        // 参数bytes：当前已完成下载的字节数
                        // 参数total：总字节数
                        // 例如
```

```
                                mFileLen = total;
                                publishProgress(bytes);
                            }
                        });
```

若此文件下载被人为或意外中断，再次下载时将检查目标路径下是否存在名为“目标文件名.vdisktemp”的文件，若判断文件名一致，则进行断点续传，将在向 Sina S3 服务器请求下载时添加 Http Header：

```
req.setHeader("Range", "bytes=" + range + "-");
req.setHeader("If-Range", "\"" + md5 + "\"");
```

以通知服务器从文件的何部分开始继续下载。

另外，调用 mFos.close();可将下载流关闭，中断下载会话，并将抛出异常

```
catch (VDiskPartialFileException e) {
// 开发者可捕获该异常并做相应处理
}
```

文件、目录操作相关接口

1.下载缩略图

使用

VDiskAPI.getThumbnail(String path, OutputStream os, ThumbSize size, ProgressListener listener) throws VDiskException

下载缩略图并将其写入本地文件中。

Parameters:

path the VDisk path to the file for which you want to get a thumbnail.

os the OutputStream to write the thumbnail to.

size the size of the thumbnail to download.

format the image format of the thumbnail to download.

listener an optional ProgressListener to receive progress updates as the thumbnail downloads, or null.

Returns:

the VDiskFileInfo for the downloaded thumbnail.

例如

```
mApi.getThumbnail(path, mFos, ThumbSize.ICON_640x480, null);
```

2.获取文件夹下的目录信息

使用

VDiskAPI.metadata(String path, String hash, boolean list, boolean includeDeleted) throws VDiskException

在当前路径指向一个文件夹时，若第三个参数为 **boolean list** 为 true，可获取到当前目录 path 下的所有文件和子目录的信息列表。此参数仅对文件夹路径生效。

Parameters:

path the VDisk path to the file or directory for which to get metadata.

hash if you previously got metadata for a directory and have it stored, pass in the returned hash. If the directory has not changed since you got the hash, a 304 VDiskServerException will be thrown. Pass in null for files or unknown directories.

list if true, returns metadata for a directory's immediate children, or just the directory entry itself if false. Ignored for files.

includeDeleted optionally gets metadata for a file at a prior rev (does not apply to folders). Use false for the latest metadata.

Returns:

a metadata Entry.

例如

```
Entry metadata = mApi.metadata(path, null, true, false);
List<Entry> contents = metadata.contents;
```

通过
for (Entry entry : contents) 可遍历得到所有文件名和文件夹名 entry.fileName()。
另外，可用 entry.isDir 字段判断是否是一个文件夹并做区别或排序操作。

3.获取文件详细信息

使用

```
Entry com.vdisk.net.VDiskAPI.metadata(String path, String hash, boolean list, boolean includeDeleted) throws VDiskException
```

在当前路径指向一个文件时，可获得该文件的详细信息。
在当前路径指向一个文件夹时，若第三个参数 boolean list 为 false，可获得该文件夹的详细信息；若 boolean list 为 true，除获得详细信息外，还可获得该目录下所有文件和子目录的信息列表，详见上一项。

Parameters:

path the VDisk path to the file or directory for which to get metadata.
hash if you previously got metadata for a directory and have it stored, pass in the returned hash. If the directory has not changed since you got the hash, a 304 VDiskServerException will be thrown. Pass in null for files or unknown directories.
list if true, returns metadata for a directory's immediate children, or just the directory entry itself if false. Ignored for files.
includeDeletedoptionally gets metadata for a file at a prior rev (does not apply to folders). Use false for the latest metadata.

Returns:

a metadata Entry.

例如

```
Entry metadata = mApi.metadata(path, null, true, false);
```

可获得

```
metadata.fileName()  文件名
metadata.size        文件大小
metadata.modified    文件修改时间
metadata.path        文件微盘路径
```

4.获取文件的历史版本

使用

```
VDiskAPI.revisions(String path, int revLimit) throws VDiskException
```

可获得指定目录下同名文件的所有历史版本列表。如果文件被执行过重命名，移动或删除操作，则可能会产生多个历史版本。开发者可通过版本号管理文件还原。

Parameters:

path the VDisk path to the file for which to get revisions (directories are not supported).

revLimit the maximum number of revisions to return. Default is 10 if you pass in 0 or less, and 1,000 is the most that will ever be returned.

Returns:

a list of metadata entries.

例如

```
List<Entry> contents = mApi.revisions(path, -1);
```

通过

for (Entry entry : contents)可遍历得到

entry.fileName()文件名

entry.rev 对应版本号

5.搜索

使用

VDiskAPI.search(String path, String query, int fileLimit, boolean includeDeleted) throws VDiskException

进行关键词搜索。

Parameters:

path the VDisk directory to search in.

query the query to search for (minimum 3 characters).

fileLimit the maximum number of file entries to return. Default is 1,000 if you pass in 0 or less, and 1,000 is the most that will ever be returned.

includeDeleted whether to include deleted files in search results.

Returns:

a list of metadata entries of matching files.

例如

```
List<Entry> result = mApi.search("/", keyword, -1, false);
```

通过

for (Entry entry : result) 可遍历得到所有符合搜索关键词的文件名和文件夹名 entry.fileName()。

另外，可用 entry.isDir 字段判断是否是一个文件夹并做区别或排序操作。

6.获取文件的下载链接

使用

VDiskAPI.media(String path, boolean ssl) throws VDiskException

获得文件的下载链接及相关信息。

Parameters:

path the VDisk path of the file for which to get a streaming link.

ssl whether the streaming URL is https or http. Some Android and other platforms won't play https streams, so false converts the link to an http link before returning it.

Returns:

a VDiskLink for streaming the file.

例如

```
VDiskLink media = mApi.media(path, false);
```

可获得

```
media.url 文件的下载地址
media.expires  下载地址的过期时间
```

7.获取用户文件和目录的操作变化记录

使用

VDiskAPI.delta(String cursor) throws VDiskException

获得一个记录，记载了某一个文件的修改操作。此记录功能仅适用于部分需要本地文件与服务器进行实时文件同步的开发者，如微盘官方 PC 客户端。

Parameters:

cursor On the first call, you should pass in null. On subsequent calls, pass in the cursor returned by the previous call.

Returns:

A single DeltaPage of results. The hasMore field will tell you whether the server has more pages of results to return. If the server doesn't have more results, you can wait a bit (say, 5 or 10 minutes) and poll again.

具体接口调用可参考 example 中的方法 `private void getDelta(final String cursor)` 进行查看。

文件编辑相关接口

1.复制

使用

VDiskAPI.copy(String fromPath, String toPath) throws VDiskException

指明目标文件地址和新文件名，进行文件拷贝操作，并返回拷贝所得文件信息。

Parameters:

fromPath the VDisk path to copy from.

toPath the full VDisk path to copy to (not just a directory).

Returns:

a metadata Entry.

例如

```
Entry metadata = mApi.copy(fromPath, toPath);
```

2.新建文件夹

使用

VDiskAPI.createFolder(String path) throws VDiskException

在指定目录位置新建一个文件夹，并返回新建的文件夹信息。

Parameters:

path the VDisk path to the new folder.

Returns:

a metadata Entry for the new folder.

例如

```
Entry metaData = mApi.createFolder(path);
```

3.删除

使用

VDiskAPI.delete(String path) throws VDiskException

进行单个文件或整个文件夹的删除操作，并返回已删除的文件信息。

Parameters:

path the VDisk path to delete.

例如

```
Entry metaData = mApi.delete(path);
```

4.**移动**

使用

```
VDiskAPI.move(String fromPath, String toPath) throws VDiskException
```

指明目标文件地址和新文件名，进行文件移动操作，并返回移动所得文件信息。

```
Parameters:
    fromPath the VDisk path to move from.
    toPath the full VDisk path to move to (not just a directory).
Returns:
    a metadata Entry.
```

例如

```
Entry metadata = mApi.move(fromPath, toPath);
```

5.还原

使用

```
VDiskAPI.restore(String path, String rev) throws VDiskException
```

可将微盘上的文件还原到某个版本号，并返回经还原后的文件信息。只要确定文件路径，在文件已经被移动甚至删除的情况下，该操作仍可以被执行。

```
Parameters:
    path the VDisk path to the file to restore.
    rev the rev to restore to (obtained from a metadata or revisions call).
Returns:
    a metadata Entry for the newly restored file.
```

例如

```
Entry metadata = mApi.restore(path, revision);
```

可获得

```
metadata.fileName() 文件名
metadata.rev        该文件的新版本号
```

分享相关接口

1.分享

使用

```
VDiskAPI.share(String path) throws VDiskException
```

可将微盘上指定文件设为公开分享状态，同时返回该文件的微盘分享链接。

```
Parameters:
    path the VDisk file path to share.
Returns:
    a Share Link for the path.
```

例如

```
String shareLink = mApi.share(path);
```

2.取消分享

使用

```
VDiskAPI.cancelShare(String path) throws VDiskException
```

取消对微盘文件的公开分享。

```
Parameters:
    path the VDisk path to cancel share.
Returns:
    the metaData of the file which is share canceled.
```

例如

```
Entry metaData = mApi.cancelShare(path);
```

3.创建拷贝引用

使用

```
VDiskAPI.createCopyRef(String sourcePath) throws VDiskException
```

获得所需文件的拷贝引用，即一个在网络服务器上唯一标识此文件的字符串。同时将此文件设置为公开共享状态。

```
Parameters:
    sourcePath The full path to the file that you want a
Returns:
    A string representation of the file pointer.
```

例如

```
CreatedCopyRef createCopyRef = mApi.createCopyRef(sourcePath);
```

并通过 createCopyRef.copyRef 获取拷贝引用字符串。

4.通过拷贝引用保存到微盘

使用

```
VDiskAPI.addFromCopyRef(String sourceCopyRef, String targetPath) throws VDiskException
```

可将拷贝引用指向的文件保存到自己的微盘中。

```
Parameters:
    sourceCopyRef The copy-ref to use as the source of the file data (comes from CreatedCopyRef.copyRef, which is created through createCopyRef()).
    targetPath The path that you want to create the file at.
Returns:
    The Entry for the new file.
```

例如

```
Entry entry = mApi.addFromCopyRef(sourceCopyRef, targetPath);
```

5.根据拷贝引用获取下载链接

使用

```
VDiskAPI.getLinkByCopyRef(String sourceCopyRef) throws VDiskException
```

获取拷贝引用对应的直接下载地址。

```
Parameters:
    sourceCopyRef file's copy Reference
```

例如

```
VDiskLink link = mApi.getLinkByCopyRef(sourceCopyRef);
```

并通过 link.url 取得下载地址。