智能媒资托管

SDK 文档

腾讯云

文档目录

SDK 文档
Android SDK
快速入门

最近更新时间：2024-10-15 20:05:41

准备工作

1. 您需要一个 Android 应用，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. 请确保您的 Android 应用目标为 API 级别 21 (Android 5.0) 或更高版本。
3. 您需要一个可以获取智能媒资托管服务访问令牌的业务服务端接口，访问令牌的相关说明请参见 生成访问令牌 。

步骤1：安装 SDK

自动集成

使用 mavenCentral 仓库

在项目级别（通常是根目录下）的 `build.gradle` 中添加：

```
repositories {
    google()
    // 增加这行
    mavenCentral()
}
```

在应用级别（通常是 app 模块下）的 `build.gradle` 中添加依赖：

```
dependencies {
    ...
    // 增加这行
    implementation 'com.qcloud.cos:smh-android:1.1.8'
}
```

关闭 beacon 上报功能

为了持续跟踪和优化 SDK 的质量，给您带来更好的使用体验，我们在 SDK 中引入了 腾讯灯塔 SDK。

> **说明**

> 腾讯灯塔只对 SMH 侧的请求性能进行监控，不会上报业务侧数据。

若是想关闭该功能，请在应用级别（通常是 app 模块下）的 `build.gradle` 中修改 smh-android 的依赖为：

```
dependencies {
    ...
    // 修改为
    implementation 'com.qcloud.cos:smh-android-nobeacon:1.1.8'
}
```

步骤2：配置权限

网络权限

SDK 需要网络权限，用于与 SMH 服务器进行通信，请在应用模块下的 `AndroidManifest.xml` 中添加如下权限声明：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

存储权限

如果您的应用场景中需要从外部存储中读写文件，请在应用模块下的 `AndroidManifest.xml` 中添加如下权限声明：

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

> ⚠ **注意**
> 在 Android 6.0（API level 23）以上，您需要在运行时动态申请存储权限。

步骤3：开始使用

实现获取访问令牌

实现一个 `SMHSimpleUser` 的子类，实现获取 libraryId 和 userSpace、请求访问令牌并返回结果的过程。

```
class MySMHSimpleUser: SMHSimpleUser() {
    override val libraryId: String
```

```
        get() = "smh3ptyc9mscifdi"
    override val userSpace: UserSpace
        get() = UserSpace(
            userId = "7",
            spaceId = "space1x8mfjgno6nyy"
            )

    override suspend fun provideAccessToken(): AccessToken {
        // 首先从您的访问令牌服务器获取包含了访问令牌信息的响应

        // 然后解析响应，获取访问令牌信息
        String token = "token"; // 访问令牌 Token
        long expiresIn = 86400;// 访问令牌的有效时长，单位为秒

        //建议返回服务器时间作为签名的开始时间，避免由于用户手机本地时间偏差过大导致请求过期
        // 返回服务器时间作为签名的起始时间
        long startTime = 1556182000L; //访问令牌有效起始时间，单位是毫秒

        // 最后返回访问令牌信息对象
        return new AccessToken(token, startTime, expiresIn);
    }
}
```

这里假设类名为 `MySMHSimpleUser` 。初始化一个实例，来给 SDK 提供访问令牌。

```
val mySMHSimpleUser: MySMHSimpleUser = MySMHSimpleUser();
```

初始化 SMHCollection

使用您提供密钥的实例 `mySMHSimpleUser` ，初始化一个 `SMHCollection` 的实例。
`SMHCollection` 提供了访问 SMH 的所有接口，建议作为**程序单例**使用，后续文档中的 SMH 均指创建的 SMHCollection 实例。

```
// 初始化 SMHCollection，获取实例
val smh: SMHCollection = SMHCollection(
            context = context,
            user = mySMHSimpleUser
        )
```

步骤4：访问 SMH 服务

例如：列出文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.list(
        //目标目录
        dir = targetDir,
        //页码
        page = 1,
        //每页拉取的数量
        pageSize = 100,
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC,
        //过滤类型
        directoryFilter = DirectoryFilter.ONLY_FILE
    )
    //文件/目录列表 其他数量等内容请查看 DirectoryContents 实体内容
    val contents = directoryContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

通用参数介绍

- LibraryId：媒体库 ID，必选参数。
- SpaceId：空间 ID，如果媒体库为单租户模式，则该参数固定为连字符（-）；如果媒体库为多租户模式，则必须指定该参数。
- AccessToken：访问令牌，必选参数。
- UserId：用户身份识别，当访问令牌对应的权限为管理员权限且申请访问令牌时的用户身份识别为空时用来临时指定用户身份，详情请参见 生成访问令牌，可选参数。
- OrganizationId：组织 ID，必选参数。
- UserToken：用户令牌，必选参数。

> **说明**
> 更多概念请参见 基本概念 。

上传与下载
上传

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件上传的 API 概览以及 SDK 示例代码。

高级上传

功能说明

该上传方法内部会自动进行快速上传、简单上传、分块上传的逻辑，生成的 uploadTask 可以供外部进行暂停、恢复、取消等。

示例代码

SDK 支持 Uri 以及输入流。下面以 Uri 为例：

```
//要上传的本地文件
val file = File.createTempFile("uploadBigMedia", ".jpg")
//获取上传任务
val uploadTask = smh.upload(
    //上传到服务端的名称
    name = "uploadBigMedia.jpg",
    //所在文件夹，默认是根目录下
    dir = Directory(),
    //本地文件 Uri
    uri = Uri.fromFile(file),
    //状态监听器
    stateListener = object : SMHStateListener {
        override fun onStateChange(request: SMHRequest, state:
SMHTransferState) {
            Log.i("Test", "onStateChange $state")
        }
    },
    //进度监听器
    progressListener = object: SMHProgressListener {
        override fun onProgressChange(request: SMHRequest, progress:
Long, target: Long) {
            Log.i("Test", "Progress change $progress/$target")
        }
```

```
    },
    //结果监听器
    resultListener = object: SMHResultListener {
        override fun onSuccess(request: SMHRequest, result: SMHResult) {
            Log.i("Test", "onSuccess")
        }
        override fun onFailure(
            request: SMHRequest,
            smhException: SMHException?,
            smhClientException: SMHClientException?
        ) {
            Log.i("Test", "onFailure $smhException and ")
        }
    }
)
launch {
    delay(5000)
    //暂停上传任务
    uploadTask.pause(true)
}
//开始上传任务
uploadTask.start()
delay(2000)
//恢复上传任务
uploadTask.resume()
```

下载

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件下载的 API 概览以及 SDK 示例代码。

高级下载

功能说明

该下载方法内部会自动进行下载到本地或流以及续传的逻辑，生成的 downloadTask 可以供外部进行暂停、恢复、取消等。

示例代码

```
//下载到本地的文件
val file = File.createTempFile("dowloadBigMedia", ".jpg")
//获取下载任务
val downloadTask = smh.download(
    //要下载的文件名称
    name = "uploadBigMedia.jpg",
    //所在文件夹，默认是根目录下
    dir = Directory(),
    //要下载到的本地文件路径
    localFullPath = file.absolutePath,
    //状态监听器
    stateListener = object : SMHStateListener {
        override fun onStateChange(request: SMHRequest, state:
SMHTransferState) {
            Log.i("testDownloadTask", "onStateChange $state")
        }
    },
    //进度监听器
    progressListener = object: SMHProgressListener {
        override fun onProgressChange(request: SMHRequest, progress:
Long, target: Long) {
            Log.i("testDownloadTask", "Progress change
$progress/$target")
        }
    },
    //结果监听器
    resultListener = object: SMHResultListener {
```

```
        override fun onSuccess(request: SMHRequest, result: SMHResult) {
            Log.i("testDownloadTask", "onSuccess")
            //下载结果
            if(result is DownloadFileResult){
                Log.i("testDownloadTask", "bytesTotal:
${result.bytesTotal}")
                Log.i("testDownloadTask", "content:
${result.content.toString()}")
                Log.i("testDownloadTask", "crc64: ${result.crc64}")
                Log.i("testDownloadTask", "key: ${result.key}")
                Log.i("testDownloadTask", "meta:
${result.meta?.entries?.joinToString()}")
            }
        }
        override fun onFailure(
            request: SMHRequest,
            smhException: SMHException?,
            smhClientException: SMHClientException?
        ) {
            Log.i("testDownloadTask", "onFailure $smhException,
$smhClientException")
        }
    }
)
launch {
    delay(5000)
    //暂停下载任务
    downloadTask.pause()
}
//开始下载任务
downloadTask.start()
delay(2000)
//恢复下载任务
downloadTask.resume()
```

搜索

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于搜索文件/目录的 API 概览以及 SDK 示例代码。

API	操作描述
搜索目录与文件	用于搜索目录与文件
继续获取搜索结果	用于继续获取搜索结果
删除搜索任务	用于删除搜索任务

搜索目录与文件

功能说明

初始化搜索，可能会返回一定量的搜索结果。

示例代码

```
try {
    val searchPartContent: SearchPartContent = smh.initSearch(
        //搜索关键字
        keyword = "keyword",
        //搜索范围
        scope = "",
        //搜索的文件类型
        searchTypes = listOf(SearchType.PDF)
    )
    //搜索结果列表 其他内容请查看 SearchPartContent 实体内容
    val contents = searchPartContent.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

查询搜索状态

功能说明

查询搜索状态。

示例代码

```
try {
    val searchPartContent: SearchPartContent = smh.searchMore(
        //搜索的 id
        searchId =
"FnBMS09MZ2V4UzZ5RVY3NmVUX0FtemchVFZaYmdvc2hTbUtYb180NnBzY3gyQToxNDA3NzE
zOTIw",
        //分页标记
        marker = 20,
    )
    //搜索结果列表 其他内容请查看 SearchPartContent 实体内容
    val contents = searchPartContent.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

完成搜索

功能说明

完成搜索。

示例代码

```
try {
    smh.deleteSearch(
        //搜索的 id
        searchId =
"FnBMS09MZ2V4UzZ5RVY3NmVUX0FtemchVFZaYmdvc2hTbUtYb180NnBzY3gyQToxNDA3NzE
zOTIw"
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

文件
创建文件链接

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
创建文件链接	创建文件链接

创建文件链接

功能说明

用于创建文件链接。

示例代码

```
//源文件名
val sourceName = "sourceName"
//目标文件名
val targetName = "targetName"
//目标文件夹
val targetDir = Directory()

try {
    val createSymLinkResult: ConfirmUpload = smh.createSymLink(
        name = targetName,
        dir = targetDir,
        sourceFileName = sourceName,
        //存在重名文件时是覆盖还是重命名，true 表示覆盖，false 表示重命名
        overrideOnNameConflict = false
    )
    //创建成功的文件链接地址
    val path = createSymLinkResult.path.joinToString("/")
} catch (e: Exception) {
    e.printStackTrace()
}
```

删除单个文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
删除文件	删除文件

删除文件

功能说明

用于删除文件。

示例代码

```
try {
    val deleteMediaResult: DeleteMediaResult? = smh.delete(
        //文件名
        name = "uploadBigMedia.jpg",
        //所在文件夹，默认是根目录下
        dir = Directory(),
        //是否永久删除，为 true 表示不放到回收站中
        permanent = false
    )
    //回收站项ID 如果未开通回收站，或者 permanent 为true 则为空
    val recycledItemId = deleteMediaResult?.recycledItemId
} catch (e: Exception) {
    e.printStackTrace()
}
```

批量删除

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量删除目录或文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量删除目录或文件	批量删除目录或文件

批量删除目录或文件

功能说明

用于批量删除目录或文件。
当项目较多以异步方式复制时，返回 HTTP 202 Accepted。
当项目较少以同步方式复制时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

示例代码

- 开始批量删除

```
try {
  //需要批量删除的文件或文件夹
  val items: List<BatchDeleteItem> = listOf(
              BatchDeleteItem(
                  //文件或目录路径
                  "filePath",
                  //是否永久删除
                  true
              ),
              BatchDeleteItem("dirPath", false)
          )
  //返回批量操作结果
  //如果是同步返回：直接从 batchResponse.result 中获取结果内容即可
  //如果是异步返回：需要调用查询任务接口获取结果内容（一般为轮询查询任务，直到查询到任务结果）
  val batchResponse: BatchResponse = smh.batchDelete(
      items = items
  )
} catch (e: Exception) {
```

```
    e.printStackTrace()
}
```

- **查询任务**
  详细使用请参考 异步处理 。

批量复制

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量复制目录或文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量复制目录或文件	批量复制目录或文件

批量复制目录或文件

功能说明

用于批量复制目录或文件。
当项目较多以异步方式复制时，返回 HTTP 202 Accepted。
当项目较少以同步方式复制时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

示例代码

- 开始批量复制

```
try {
  //需要批量复制的文件或文件夹
  val items: List<BatchCopyItem> = listOf(
      BatchCopyItem(
          //被复制的源目录、相簿或文件路径
          "fromPath",
          //目标目录、相簿或文件路径
          "toPath",
          //文件名冲突时的处理方式
          ConflictStrategy.RENAME
      ),
      BatchCopyItem("fromPath", "toPath", ConflictStrategy.RENAME)
  )
  //返回批量操作结果
  //如果是同步返回：直接从 batchResponse.result 中获取结果内容即可
  //如果是异步返回：需要调用查询任务接口获取结果内容（一般为轮询查询任务，直到查询到任务结果）
  val batchResponse: BatchResponse = smh.batchCopy(
      items = items
```

```
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

- **查询任务**
  详细使用请参考 **异步处理** 。

批量移动

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量移动目录或文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量移动目录或文件	批量移动目录或文件

批量移动目录或文件

功能说明

用于批量移动目录或文件。
当项目较多以异步方式复制时，返回 HTTP 202 Accepted。
当项目较少以同步方式复制时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

示例代码

- 开始批量移动

```
try {
  //需要批量移动的文件或文件夹
  val items: List<BatchMoveItem> = listOf(
      BatchMoveItem(
          //被移动的源目录、相簿或文件路径
          "fromPath",
          //目标目录、相簿或文件路径
          "toPath",
          //文件名冲突时的处理方式
          ConflictStrategy.RENAME,
          //是否移动文件夹权限
          true
      ),
      BatchMoveItem("fromPath", "toPath", ConflictStrategy.RENAME,
false)
  )
  //返回批量操作结果
  //如果是同步返回：直接从 batchResponse.result 中获取结果内容即可
```

```
    //如果是异步返回：需要调用查询任务接口获取结果内容（一般为轮询查询任务，直到查询
到任务结果）
    val batchResponse: BatchResponse = smh.batchMove(
        items = items
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

- **查询任务**
  详细使用请参考 异步处理 。

检查文件状态

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于检查文件状态的 API 概览以及 SDK 示例代码。

API	操作描述
检查文件状态	检查文件状态

检查文件状态

功能说明

用于检查文件状态。

示例代码

```
//目标文件名
val targetName = "targetName"
//目标文件夹
val targetDir = Directory()

try {
    val headFileContent: HeadFileContent = smh.headFile(
        name = targetName,
        dir = targetDir
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取文件信息

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于获取文件信息的 API 概览以及 SDK 示例代码。

API	操作描述
获取文件下载链接和信息	获取文件下载链接和信息

获取文件信息

功能说明

用于获取文件信息。

示例代码

```
//目标文件名
val targetName = "targetName"
//目标文件夹
val targetDir = Directory()

try {
    val fileInfo: FileInfo = smh.getFileInfo(
        name = targetName,
        dir = targetDir,
        //历史版本号
        historyId = 123456,
        //用途（用于标记该链接的使用场景）
        purpose = Purpose.PREVIEW
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取文件链接

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于获取文件下载链接、文档预览链接、缩略图链接的 API 概览以及 SDK 示例代码。

API	操作描述
获取文档预览	获取文档预览链接
获取文件下载链接	获取文件下载链接
获取缩略图	获取照片/视频封面缩略图链接

获取文档预览

功能说明

用于获取文档预览链接。

示例代码

```
try {
    val previewAccessUrl: String = smh.getPreviewAccessUrl(
        //文件路径
        filePath = "filePath",
        //历史版本号
        historyId = 123456,
        //用途（用于标记该链接的使用场景）
        purpose = Purpose.PREVIEW
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取文件下载链接

功能说明

用于获取文件下载链接。

示例代码

```
try {
    val downloadAccessUrl: String = smh.getDownloadAccessUrl(
        //文件路径
        filePath = "filePath",
        //历史版本号
        historyId = 123456,
        //文件路径是否已经 url 编码
        encode = true
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取缩略图

功能说明

用于获取缩略图链接。

示例代码

- 示例1：

```
try {
  val thumbnailUrl: String = smh.getThumbnailAccessUrl(
      //文件名
      name = "name",
      //目录名
      dir = Directory(),
      //历史版本号
      historyId = 123456,
      //生成的预览图尺寸
      size = 100,
      //等比例缩放百分比，不传 Size 时生效
      scale = 80,
      //缩放宽度，不传高度时，高度按等比例缩放，不传 Size 和 Scale 时生效；
      widthSize = 100,
      //缩放高度，不传宽度时，宽度按等比例缩放，不传 Size 和 Scale 时生效；
      heightSize = 100,
      //用途(用于标记该链接的使用场景)
      purpose = Purpose.PREVIEW
  )
} catch (e: Exception) {
```

```
    e.printStackTrace()
}
```

- 示例2：

```
try {
    val thumbnailResult: ThumbnailResult = smh.getThumbnail(
        //文件名
        name = "name",
        //目录名
        dir = Directory(),
        //生成的预览图尺寸
        size = 100,
        //等比例缩放百分比，不传 Size 时生效
        scale = 80,
        //缩放宽度，不传高度时，高度按等比例缩放，不传 Size 和 Scale 时生效；
        widthSize = 100,
        //缩放高度，不传宽度时，宽度按等比例缩放，不传 Size 和 Scale 时生效；
        heightSize = 100,
        //帧数，针对 gif 的降帧处理
        frameNumber = 6,
        //用途(用于标记该链接的使用场景)
        purpose = Purpose.PREVIEW
    )
    val thumbnailUrl = thumbnailResult.location
} catch (e: Exception) {
    e.printStackTrace()
}
```

重命名文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
重命名或移动文件	重命名或移动文件

重命名或移动文件

功能说明

用于重命名或移动文件。

示例代码

```
//源文件名
val sourceName = "sourceName"
//源文件夹
val sourceDir = Directory()
//目标文件名
val targetName = "targetName"
//目标文件夹
val targetDir = Directory()

try {
    val renameFileResponse: RenameFileResponse = smh.renameFile(
        targetName = targetName,
        targetDir = targetDir,
        sourceName = sourceName,
        sourceDir = sourceDir,
        //文件名冲突时策略
        conflictStrategy = ConflictStrategy.RENAME
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

目录或相簿
创建文件夹

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
创建目录或相簿	创建目录或相簿

创建目录或相簿

功能说明

用于创建目录或相簿。

示例代码

```
//目标文件夹
val targetDir = Directory()
try {
    val createDirectoryResult: CreateDirectoryResult =
smh.createDirectory(
        dir = targetDir
    )
    val creationTime = createDirectoryResult.creationTime
} catch (e: Exception) {
    e.printStackTrace()
}
```

列出内容

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于列出目录或相簿内容的 API 概览以及 SDK 示例代码。

API	操作描述
列出目录或相簿	用于列出目录或相簿内容

列出目录或相簿

功能说明

用于列出目录或相簿内容。

示例代码

列出文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.list(
        //目标目录
        dir = targetDir,
        //页码
        page = 1,
        //每页拉取的数量
        pageSize = 100,
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC,
        //过滤类型
        directoryFilter = DirectoryFilter.ONLY_FILE
    )
    //文件/目录列表 其他数量等内容请查看 DirectoryContents 实体内容
    val contents = directoryContents.contents
} catch (e: Exception) {
    e.printStackTrace()
```

```
}
```

通过 marker + limit 的方式列出文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.listWithMarker(
        //目标目录
        dir = targetDir,
        //用于顺序列出分页的标识
        marker = 100,
        //用于顺序列出分页时本地列出的项目数限制
        limit = 100,
        //当前目录的 ETag
        eTag = "eTag",
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC,
        //过滤类型
        directoryFilter = DirectoryFilter.ONLY_FILE
    )
    //文件/目录列表 其他数量等内容请查看DirectoryContents实体内容
    val contents = directoryContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

通过 offset + limit 的方式来列出文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.listWithOffset(
        //目标目录
        dir = targetDir,
        //文件偏移量
        offset = 100,
        //列出的数量
        limit = 100,
        //排序方式
```

```
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC,
        //过滤类型
        directoryFilter = DirectoryFilter.ONLY_FILE
    )
    //文件/目录列表 其他数量等内容请查看 DirectoryContents 实体内容
    val contents = directoryContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

列出所有的文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.listAll(
        //目标目录
        dir = targetDir,
        //每页拉取的数量
        pageSize = 100,
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC,
        //过滤类型
        directoryFilter = DirectoryFilter.ONLY_FILE
    )
    //文件/目录列表 其他数量等内容请查看 DirectoryContents 实体内容
    val contents = directoryContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

通过 marker + limit 的方式列出所有的文件列表

```
//目标文件夹
val targetDir = Directory()
try {
    val directoryContents: DirectoryContents = smh.listAllWithMarker(
        //目标目录
```

```
            dir = targetDir,
            //每次拉取的数量
            limit = 100
        )
        //文件/目录列表 其他数量等内容请查看 DirectoryContents 实体内容
        val contents = directoryContents.contents
    } catch (e: Exception) {
        e.printStackTrace()
    }
```

共享文件夹

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于获取我共享的文件夹的 API 概览以及 SDK 示例代码。

API	操作描述
获取我共享的文件夹	用于获取我共享的文件夹

获取我共享的文件夹

功能说明

用于获取我共享的文件夹。

示例代码

列出文件夹列表

```
try {
    val authorizedContent: AuthorizedContent =
smh.getMyAuthorizedDirectory(
        //页码
        page = 1,
        //每页拉取的数量
        pageSize = 100,
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC
    )
    //文件夹列表 其他数量等内容请查看 AuthorizedContent 实体内容
    val contents = authorizedContent.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

通过 marker + limit 的方式列出

```
try {
```

```
    val authorizedContent: AuthorizedContent =
smh.getMyAuthorizedDirectoryWithMarker(
        //用于顺序列出分页的标识
        marker = 100,
        //用于顺序列出分页时本地列出的项目数限制
        limit = 100,
        //当前目录的 ETag
        eTag = "eTag",
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC
    )
    //文件夹列表 其他数量等内容请查看 AuthorizedContent 实体内容
    val contents = authorizedContent.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

删除文件夹

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
删除目录或相簿	删除目录或相簿

删除目录或相簿

功能说明

用于删除目录或相簿。

示例代码

```
//目标文件夹
val targetDir = Directory()
try {
    val checkSuccess: Boolean = smh.deleteDirectory(
        dir = targetDir
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取目录信息

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于查看目录或相簿的 API 概览以及 SDK 示例代码。

API	操作描述
查看目录或相簿	查看目录或相簿

查看目录或相簿

功能说明

用于查看目录或相簿信息。

示例代码

```
//目标文件夹
val targetDir = Directory()

try {
    val directoryInfo: DirectoryInfo = smh.getDirectoryInfo(
        dir = targetDir
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

获取相簿封面

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于查看目录或相簿的 API 概览以及 SDK 示例代码。

API	操作描述
获取相簿封面	获取相簿封面

获取相簿封面链接

功能说明

用于获取相簿封面链接。

示例代码

```
try {
    val albumCoverUrl: String? = smh.getAlbumCoverUrl(
        //相簿名，分相簿媒体库必须指定该参数，不分相簿媒体库不能指定该参数
        albumName = "albumName"
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

重命名文件夹

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
重命名或移动目录或相簿	重命名或移动目录或相簿

重命名或移动目录或相簿

功能说明

用于重命名或移动目录或相簿。

示例代码

```
//源文件夹
val sourceDir = Directory()
//目标文件夹
val targetDir = Directory()

try {
    val renameFileResponse: RenameFileResponse = smh.renameDirectory(
        target = targetDir,
        source = sourceDir,
        //文件名冲突时策略
        conflictStrategy = ConflictStrategy.RENAME
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

角色权限
获取角色列表

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于用于获取角色列表的 API 概览以及 SDK 示例代码。

API	操作描述
获取角色列表	用于获取角色列表

获取角色列表

功能说明

用于获取角色列表。

示例代码

```
try {
    //角色列表，不同的角色对应不同的权限
    val roles: List<Role> = smh.getRoleList()
} catch (e: Exception) {
    e.printStackTrace()
}
```

回收站
回收站列表

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于列出回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
列出回收站项目	用于列出回收站项目

列出回收站项目

功能说明

用于列出回收站项目。

示例代码

列出回收站列表

```
try {
    val recycledContents: RecycledContents = smh.listRecycled(
        //页码
        page = 1,
        //每页拉取的数量
        pageSize = 100,
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC
    )
    //回收站列表 其他数量等内容请查看 RecycledContents 实体内容
    val contents = recycledContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

通过 marker + limit 的方式列出

```
try {
    val recycledContents: RecycledContents = smh.listRecycledWithMarker(
        //用于顺序列出分页的标识
        marker = 100,
        //用于顺序列出分页时本地列出的项目数限制
        limit = 100,
        //当前目录的 ETag
        eTag = "eTag",
        //排序方式
        orderType = OrderType.NAME,
        //排序方向
        orderDirection = OrderDirection.ASC
    )
    //回收站列表 其他数量等内容请查看 RecycledContents 实体内容
    val contents = recycledContents.contents
} catch (e: Exception) {
    e.printStackTrace()
}
```

删除回收站文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于删除回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
删除回收站项目	删除回收站项目
批量删除回收站项目	批量删除回收站项目
清空回收站	清空回收站

删除回收站项目

功能说明

用于删除回收站项目。

示例代码

```
try {
    val checkSuccess: Boolean = smh.deleteRecycledItem(
        //回收站文件 id
        itemId = 123456
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

批量删除回收站项目

功能说明

用于批量删除回收站项目。

示例代码

```
try {
    val checkSuccess: Boolean = smh.deleteRecycledItems(
        //回收站文件 id 列表
```

```
        itemIds = listOf(123456, 123459)
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

清空回收站

功能说明

用于清空回收站。

示例代码

```
try {
    val checkSuccess: Boolean = smh.clearRecycledItem()
} catch (e: Exception) {
    e.printStackTrace()
}
```

恢复回收站文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于删除回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
恢复回收站项目	恢复回收站项目
批量恢复回收站项目	批量恢复回收站项目

恢复回收站项目

功能说明

用于恢复回收站项目。

示例代码

```
try {
    //最终的文件路径
    val path: String? = smh.restoreRecycledItem(
        //回收站文件 ID
        itemId = 123456
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

批量恢复回收站项目

功能说明

用于批量恢复回收站项目。
执行成功
当项目较多以异步方式复制时，返回 HTTP 202 Accepted；
当项目较少以同步方式复制时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）；

示例代码

- 开始批量恢复

```
try {
  //返回批量操作结果
  //如果是同步返回：直接从 batchResponse.result 中获取结果内容即可
  //如果是异步返回：需要调用查询任务接口获取结果内容（一般为轮询查询任务，直到查询到任务结果）
  val batchResponse: BatchResponse = smh.restoreRecycledItems(
      //回收站文件 id 列表
      itemIds = listOf(123456, 123459)
  )
} catch (e: Exception) {
  e.printStackTrace()
}
```

- 查询任务

  详细使用请参考 异步处理 。

异步处理

最近更新时间：2024-10-15 20:05:41

简介

本文档提供用于查询耗时任务执行情况的 API 概览以及 SDK 示例代码。

API	操作描述
查询任务	用于查询耗时任务执行情况

查询任务

功能说明

用于查询耗时任务执行情况。

- 任务的具体返回请参阅会产生异步任务的接口说明，部分接口会根据任务耗时情况返回同步或异步结果，此时异步结果通常与同步结果的格式保持一致。
- 仅能查询到任务结束时间在最近30天的任务，更早期的任务无法查询。
- 一般需要轮询调用该接口，直到查询到任务结果。

示例代码

```
try {
    //需要查询的任务ID 列表
    val taskIds: List<Long> = listOf()
    //返回查询结果
    val batchResponses: List<BatchResponse> = smh.queryTasks(
        taskIds = taskIds
    )
} catch (e: Exception) {
    e.printStackTrace()
}
```

Java 调用

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于使用 Java 语言调用 SMH SDK 的说明。

SMHCollectionFuture

SMHCollectionFuture 是 SMH 资源库封装类，提供适合 Java8 CompletableFuture 风格的 API。SMHCollection 中所有方法在 CompletableFuture 中都能找到对应的 CompletableFuture 风格的方法，它们名称、参数、返回值（ `CompletableFuture<XXX>` ）均一样。

示例代码

```
//生成 SMHCollectionFuture 示例
Context context;
SMHCollectionFuture smh = new SMHCollection(
        context,
        new MySMHSimpleUser()
).future();

//获取目录列表
CompletableFuture<DirectoryContents> cf = smh.list(1, 50);
//阻塞获取结果
// try {
//     DirectoryContents directoryContents = cf.get();
// } catch (Throwable e){
//     e.printStackTrace();
// }

//异步获取结果
// 如果执行成功:
cf.thenApply((result) -> {
    DirectoryContents directoryContents = result;
    return result;
});
// 如果执行异常:
cf.exceptionally((e) -> {
    e.printStackTrace();
    return null;
```

```
});
```

自定义 DNS

最近更新时间：2024-12-10 16:58:11

简介

本文档提供关于 SMH SDK 配置自定义 DNS 的说明。

自定义 DNS

一般用来接入 HTTP DNS，实现域名防劫持，DNS 解析加速的效果。
通过 SMHService 的静态成员 dnsFetch 进行 DNS 设置。

```
SMHService.dnsFetch = QCloudHttpClient.QCloudDnsFetch { hostname ->
        //根据hostname获取对应的IP列表
        //返回对应的InetAddress列表
        return inetAddressList;
    }
}
```

接入 HTTP DNS

以下使用腾讯云 HTTP DNS 作为示例。
1. 初始化腾讯云 HTTP DNS SDK，请参见 腾讯云 HTTP DNS 官方文档 。
2. 通过 SMHService.dnsFetch 配置腾讯云 HTTP DNS，实现 SMH SDK 接入 HTTP DNS。

```
SMHService.dnsFetch = QCloudHttpClient.QCloudDnsFetch { hostname ->
    val ips = MSDKDnsResolver.getInstance().getAddrByName(hostname)
    val ipArr = ips.split(";")
    if (ipArr.isEmpty()) {
        emptyList<InetAddress>()
    } else {
        val inetAddressList: MutableList<InetAddress> =
ArrayList(ipArr.size)
        for (ip in ipArr) {
            if ("0" == ip) {
                continue
            }
            try {
                val inetAddress = InetAddress.getByName(ip)
                inetAddressList.add(inetAddress)
            } catch (ignored: UnknownHostException) {
```

```
                }
            }
            inetAddressList
        }
    }
```

iOS SDK
快速入门

最近更新时间：2024-10-15 20:05:41

准备工作

1. 您需要一个 iOS 应用，这个应用可以是您现有的工程，也可以是您新建的一个空的工程。
2. 请确保您的 iOS 应用目标为 iOS 9及以上。

步骤1：安装 SDK

podfile 文件中增加 `pod "QCloudCOSSMH/Api"`，终端执行 `pod install`

步骤2：开始使用

导入头文件

```
#import <QCloudCOSSMHApi.h>
```

初始化 SMH 服务并实现获取 accessToken 协议

> **说明：**
> - User 模块不需要 accesstoken，若仅使用 User，可以跳过此步骤。
> - 集成 API 和 User 时，获取 accessToken 的方式。

API 模块接口在发出请求时需要携带 accesstoken，所以需要实现 QCloudSMHAccessTokenProvider 协议，在该协议中获取包含 accesstoken 以及 spaceid 等信息的 QCloudSMHSpaceInfo 对象通过参数 continueBlock 回调给 SDK。

```
- (void)accessTokenWithRequest:(QCloudSMHBizRequest *)request
                    urlRequest:(NSURLRequest *)urlRequst
                     compelete:
(QCloudSMHAuthentationContinueBlock)continueBlock {

        // 首先从您的访问令牌服务器获取包含了访问令牌信息的响应
        QCloudSMHSpaceInfo * spaceInfo = [QCloudSMHSpaceInfo new];
        spaceInfo.accessToken = @""; // 访问令牌 Token
        spaceInfo.expiresIn = @""; // 访问令牌的有效时长，单位为秒
```

```
        spaceInfo.libraryId = @""; //
        spaceInfo.spaceId = @"";
        continueBlock(sapceInfo, nil);
}
```

SDK 提供了一个 QCloudSMHAccessTokenFenceQueue 的脚手架，实现对 accessToken 的缓存与复用。脚手架在密钥过期之后会重新调用该协议的方法来重新获取新的密钥，直到该密钥过期时间大于设备的当前时间。

> **说明：**
> 建议把初始化过程放在 AppDelegate 或者程序单例中。

使用脚手架您需要实现 `QCloudAccessTokenFenceQueueDelegate` 协议。

1. 初始化脚手架。

```
@property (nonatomic) QCloudSMHAccessTokenFenceQueue *fenceQueue;

self.fenceQueue = [QCloudSMHAccessTokenFenceQueue new];
self.fenceQueue.delegate = self;
```

2. 实现 `QCloudAccessTokenFenceQueueDelegate` 。

```
- (void)fenceQueue:(QCloudSMHAccessTokenFenceQueue *)queue
           request:(QCloudSMHBizRequest *)request
requestCreatorWithContinue:
(QCloudAccessTokenFenceQueueContinue)continueBlock {
    // 首先从您的访问令牌服务器获取包含了访问令牌信息的响应
    QCloudSMHSpaceInfo * spaceInfo = [QCloudSMHSpaceInfo new];
    spaceInfo.accessToken = @""; // 访问令牌 Token
    spaceInfo.expiresIn = @""; // 访问令牌的有效时长，单位为秒
    spaceInfo.libraryId = @""; //
    spaceInfo.spaceId = @"";
    continueBlock(sapceInfo, nil);
}

// 在该方法中使用脚手架进行请求sdk内部缓存的accesstoken并使用continueBlock回调
给sdk，若
// sdk内缓存的accesstoken过期或没有，则跳转到上面方法中进行请求最新的
accesstoken回调给sdk并缓存。
- (void)accessTokenWithRequest:(QCloudSMHBizRequest *)request
                    urlRequest:(NSURLRequest *)urlRequst
```

```
                    compelete:
(QCloudSMHAuthentationContinueBlock)continueBlock {
    [self.fenceQueue performRequest:request
                          withAction:^(QCloudSMHSpaceInfo *_Nonnull
accessToken, NSError *_Nonnull error) {
        if (error) {
            continueBlock(nil, error);
        } else {
            continueBlock(accessToken, nil);
        }
    }];
}
```

> **说明：**
> 若仅需要 API 模块，则需要一个可以获取智能媒资托管服务访问令牌的业务服务端接口，访问令牌的相关说明请参见 生成访问令牌 。

实现 QCloudSMHAccessTokenProvider 协议，在该协议中获取 accesstoken 并包装成一个 QCloudSMHSpaceInfo 对象通过参数 continueBlock 回调给 SDK。

步骤3：访问 SMH 服务

例如：列出文件列表。

```
QCloudSMHListContentsRequest *req = [QCloudSMHListContentsRequest new];
// 用户所在空间 ID
req.spaceId = @"spaceId";
// 用户所在 libraryid
req.libraryId = @"libraryId";

// 目录路径或相簿名，对于多级目录，使用斜杠(/)分隔，例如 foo/bar；对于根目录，该参数
留空;
req.dirPath = @"dirpath";
[req setFinishBlock:^(QCloudSMHContentListInfo *_Nullable result,
NSError *_Nullable error) {
    // result 文件列表数据
    // error 报错信息
}];
// 发起请求
[[QCloudSMHService defaultSMHService] listContents:req];
```

> **说明：**
> SMH SDK 提供了自定义域名的功能，若业务有自研后台，可以使用如下方式更改访问域名，按需配置。

```
// 配置发布域名
[QCloudSMHBaseRequest setBaseRequestHost:@"releasehost"
targetType:QCloudECDTargetRelease];
// 配置预发布域名
[QCloudSMHBaseRequest setBaseRequestHost:@"previewhost"
targetType:QCloudECDTargetPreview];
// 配置测试域名
[QCloudSMHBaseRequest setBaseRequestHost:@"testhost"
targetType:QCloudECDTargetTest];
// 配置开发域名
[QCloudSMHBaseRequest setBaseRequestHost:@"devhost"
targetType:QCloudECDTargetDevelop];
// 设置当前模式 开发、测试、预发布、发布
[QCloudSMHBaseRequest setTargetType:QCloudECDTarget];
```

通用参数介绍

- libraryId：媒体库 ID，必选参数。
- spaceId：空间 ID，如果媒体库为单租户模式，则该参数固定为连字符（-）；如果媒体库为多租户模式，则必须指定该参数。
- accessToken：访问令牌，必选参数。

> **说明：**
> - 更多概念请参见 基本概念 。
> - 查看全部接口文档请参见 全部文档 。

上传与下载
上传

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件上传的 API 概览以及 SDK 示例代码。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考 。

高级上传

功能说明

该上传方法内部会自动进行快速上传、简单上传、分块上传的逻辑。

开始上传或继续上传

```
QCloudCOSSMHUploadObjectRequest *uploadReq =
[QCloudCOSSMHUploadObjectRequest new];
// confirmKey 若设置则视为继续上传，否则为新上传任务。
// 从 getConfirmKey 回调获取。
uploadReq.confirmKey = @"confirmKey";
// 设置要上传的目录所在 libraryId;
uploadReq.libraryId = @"libraryId";
// 设置要上传的目录所在 libraryId;
uploadReq.spaceId = @"spaceId";
// 上传的文件 URL 或者 data
uploadReq.body = @"uploadBody";
uploadReq.uploadPath = @"目标路径";
// confirmKey 回调
uploadReq.getConfirmKey = ^(NSString *_Nullable confirmKey) {
    // confirmKey 在进行断点续传时需要，业务需要保存
};
// 上传进度回调
[uploadReq setSendProcessBlock:^(int64_t bytesSent, int64_t
totalBytesSent, int64_t totalBytesExpectedToSend) {

}];
// 上传结果回调
```

```
[uploadReq setFinishBlock:^(QCloudSMHContentInfo *result, NSError
*error) {

}];
// 发起上传
[[QCloudSMHService defaultSMHService] uploadObject:uploadReq];
```

暂停

上传时创建的 request 执行 cancel 方法。

```
[uploadReq cancel];
```

取消上传

上传时创建的 request 执行 abort 方法，可以在结束回调中做一些清理工作。

```
[uploadReq abort:^(id outputObject, NSError *error) {
    // 结束回调
}];
```

说明：
若要实现 App 被 kill 之后再次打开继续续传，则需要业务端将 confirmkey 以及本次上传任务对应的 body、libraryId、spaceId、uploadPath 进行持久化，再次进入 App 时重新创建 request 并开始上传。

下载

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件下载的 API 概览以及 SDK 示例代码。

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参考 SDK API 参考 。

高级下载

功能说明

该下载方法内部会自动进行下载到本地、暂停、续传、取消的逻辑。

开始下载或继续下载

```
QCloudCOSSMHDownloadObjectRequest *req =
[QCloudCOSSMHDownloadObjectRequest new];

// 文件全路径
req.filePath = @"filepath";
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 该选项设置为 YES 后，在下载完成后会比对 COS 上储存的文件 crc64和下载到本地的文件
crc64
目前默认开启。
req.enableCRC64Verification = YES;
// 指定是否使用分块及续传下载，默认为 YES。
req.resumableDownload = YES;
// 本地下载路径
req.downloadingURL = cto.tempFileURL;
// 进度回调
[req setDownProcessBlock:^(int64_t bytesDownload, int64_t
totalBytesDownload, int64_t totalBytesExpectedToDownload) {

}];
// 完成回调
```

```
[req setFinishBlock:^(id outputObject, NSError *error) {

}];
[[QCloudSMHService defaultSMHService] smhDownload:req];
```

暂停

```
[request cancel];
```

删除或取消

```
[request remove];
```

批量操作
批量恢复回收站项目

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量恢复回收站文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量恢复回收站文件	批量恢复回收站文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

批量恢复回收站文件

功能说明

- 用于批量恢复目录或文件
- 当项目较多以异步方式恢复时，返回 HTTP 202 Accepted。
- 当项目较少以同步方式恢复时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

批量恢复回收站文件示例代码

开始批量恢复

```
QCloudSMHBatchRestoreRecycleObjectReqeust *req =
[QCloudSMHBatchRestoreRecycleObjectReqeust new];
req.priority = self.priority;
req.libraryId = self.libraryId;
req.spaceId = self.spaceId;
req.recycledItemIds = self.batchInfos;
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError * _Nullable
error) {
  //如果是同步任务，从 http 的状态码中获取任务的状态
  QCloudSMHBatchTaskStatus status =
QCloudSMHBatchTaskStatusTypeFromStatus([result
__originHTTPURLResponse__].statusCode);
```

```
    result.status = status;
    if(status != QCloudSMHBatchTaskStatusWating || error){
        // 当任务状态非等待或者 有 error 时 结束
        if(self.finishBlock){
            self.finishBlock(result, error);
        }
    }else{
        // 使用返回的 taskid 进行轮询任务状态
        result.taskId;
    }

}];
[[QCloudSMHService defaultSMHService]batchRestoreRecycleObject:req];
```

查询任务状态

这里需要进行轮询任务状态，直到查询到任务结果。

```
QCloudGetTaskStatusRequest *req = [QCloudGetTaskStatusRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
// 上一步返回的 taskId
req.taskIdList = @[taskId];
[req setFinishBlock:^(NSArray * _Nonnull result, NSError * _Nonnull
error) {

}];
[[QCloudSMHService defaultSMHService] getTaskStatus:req];
```

高级批量恢复回收站文件示例代码

该接口封装了批量恢复回收站文件以及轮询任务状态，setFinishBlock 直接返回最终任务结果，无需手动查询任务状态。

```
QCloudSMHRestoreObjectRequest *req = [QCloudSMHRestoreObjectRequest
new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";

req.batchInfos = @[@"1"];
```

```
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] restoreObject:req];
```

批量删除回收站项目

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量删除回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
删除回收站项目	删除回收站项目

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

批量删除回收站项目

功能说明

用于批量删除回收站项目。

示例代码

```
QCloudSMHBatchDeleteRecycleObjectReqeust *req =
[QCloudSMHBatchDeleteRecycleObjectReqeust new];
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
// 回收站项目 ID，必选参数；
req.recycledItemIds = @[@"1",@"2"];
[req setFinishBlock:^(id _Nullable outputObject, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] batchDeleteRecycleObject:req];
```

批量删除文件

最近更新时间：2022-08-24 15:53:33

简介

本文档提供关于批量删除文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量删除文件	批量删除文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

批量删除

功能说明

用于批量删除目录或文件
当项目较多以异步方式删除时，返回 HTTP 202 Accepted。
当项目较少以同步方式删除时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

批量删除实例代码

- 开始批量删除

```
QCloudSMHBatchDeleteRequest *req = [QCloudSMHBatchDeleteRequest new];
req.libraryId = @"libraryId";
req.spaceId = @"spaceId";
req.userId = @"userId";
req.spaceOrgId = @"spaceOrgId";
req.batchInfos = @[];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *
_Nullable error) {
  // 如果是同步任务，从 http 的状态码中获取任务的状态
  QCloudSMHBatchTaskStatus status =
QCloudSMHBatchTaskStatusTypeFromStatus([result
__originHTTPURLResponse__].statusCode);
  result.status = status;
  if(status != QCloudSMHBatchTaskStatusWating || error){
      // 当任务状态非等待或者 有 error 时 结束
```

```
        if(self.finishBlock){
            self.finishBlock(result, error);
        }
    }else{
        // 使用返回的 taskid 进行轮询任务状态
        self.taskId = result.taskId;
    }
}];
[[QCloudSMHService defaultSMHService]batchDelete:req];
```

- **查询任务状态**
  这里需要进行轮询任务状态，直到查询到任务结果。

```
QCloudGetTaskStatusRequest *req = [QCloudGetTaskStatusRequest new];
req.spaceId = @"spaceId";
req.spaceOrgId = @"spaceOrgId";
req.libraryId = @"libraryId";
req.userId = @"userId";
// 上一步返回的 taskId
req.taskIdList = @[taskId];
[req setFinishBlock:^(NSArray * _Nonnull result, NSError * _Nonnull
error) {

}];
[[QCloudSMHService defaultSMHService] getTaskStatus:req];
```

高级批量删除示例代码

该接口封装了批量删除以及轮询任务状态，setFinishBlock 直接返回最终任务结果，无需手动查询任务状态。

```
QCloudSMHDeleteObjectRequest *req = [QCloudSMHDeleteObjectRequest new];
req.spaceId = @"spaceId";
req.spaceOrgId = @"spaceOrgId";
req.libraryId = @"libraryId";
req.userId = @"userId";

QCloudSMHBatchDeleteInfo *info = [QCloudSMHBatchDeleteInfo new];
// 文件路径
info.path = @"totalPath";
req.batchInfos = @[info];
```

```
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] deleteObject:req];
```

批量复制文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量复制文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量复制文件	批量复制文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

批量复制文件

功能说明

- 用于批量复制目录或文件，不支持跨空间复制。
- 当项目较多以异步方式复制时，返回 HTTP 202 Accepted。
- 当项目较少以同步方式复制时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

批量复制文件代码

开始批量复制

```
QCloudSMHBatchCopyRequest *req = [QCloudSMHBatchCopyRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
// 实现保存网盘功能时需要该参数
req.shareAccessToken = @"shareAccessToken";
QCloudSMHBatchCopyInfo *info = [QCloudSMHBatchCopyInfo new];
// 原路径
info.from = @"from";
// 目标路径
info.to = @"target";
// fromLibraryId fromSpaceId 参数用来实现保存网盘功能。需要和 shareAccessToken
一起使用。
info.fromLibraryId = @"fromLibraryId";
info.fromSpaceId = @"fromSpaceId";
// 是否移动权限
```

```
info.moveAuthority = YES;
// 文件名与目标路径冲突时解决策略
info.conflictStrategy = QCloudSMHConflictStrategyEnumRename;
req.batchInfos = @[info];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError * _Nullable
error) {
  //如果是同步任务，从 http 的状态码中获取任务的状态
  QCloudSMHBatchTaskStatus status =
QCloudSMHBatchTaskStatusTypeFromStatus([result
__originHTTPURLResponse__].statusCode);
  result.status = status;
  if(status != QCloudSMHBatchTaskStatusWating || error){
      // 当任务状态非等待或者有 error 时结束
      if(self.finishBlock){
          self.finishBlock(result, error);
      }
  }else{
      // 使用返回的 taskid 进行轮询任务状态
      result.taskId;
  }
}];
[[QCloudSMHService defaultSMHService]batchCopy:req];
```

查询任务状态

这里需要进行轮询任务状态，直到查询到任务结果。

```
QCloudGetTaskStatusRequest *req = [QCloudGetTaskStatusRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
// 上一步返回的 taskId
req.taskIdList = @[taskId];
[req setFinishBlock:^(NSArray * _Nonnull result, NSError * _Nonnull
error) {

}];
[[QCloudSMHService defaultSMHService] getTaskStatus:req];
```

高级批量复制示例代码

该接口封装了批量复制以及轮询任务状态，setFinishBlock 直接返回最终任务结果，无需手动查询任务状态。

```
QCloudSMHCopyObjectRequest *req = [QCloudSMHCopyObjectRequest new];
req.libraryId = @"libraryId";
req.spaceId = @"spaceId";
// 保存网盘时需要 shareAccessToken
req.shareAccessToken = @"shareAccessToken";

QCloudSMHBatchCopyInfo *info = [QCloudSMHBatchCopyInfo new];
// 源路径
info.from = @"fromPath";
// 目标路径
info.to = @"toPath";
info.conflictStrategy = QCloudSMHConflictStrategyEnumRename;
// 实现保存网盘是需要 shareAccessToken fromSpaceId fromLibraryId
if (shareAccessToken) {
    info.fromSpaceId = @"fromSpaceId";
    info.fromLibraryId = @"fromLibraryId";
}
req.batchInfos = @[info];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] copyObject:req];
```

批量移动文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于批量重命名或移动目录或文件的 API 概览以及 SDK 示例代码。

API	操作描述
批量重命名或移动目录或文件	批量重命名或移动目录或文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

批量重命名或移动目录或文件

功能说明

- 用于批量重命名或移动目录或文件
- 当项目较多以异步方式移动时，返回 HTTP 202 Accepted。
- 当项目较少以同步方式移动时，返回 HTTP 200 OK（全部执行成功）或 HTTP 207 Multi-Status（存在部分或全部执行失败）。

批量移动实例代码

开始批量移动

```
QCloudSMHBatchMoveRequest *req = [QCloudSMHBatchMoveRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
QCloudSMHBatchMoveInfo *info = [QCloudSMHBatchMoveInfo new];
// 原路径
info.from = @"from";
// 目标路径
info.to = @"target";
// 是否移动权限
info.moveAuthority = YES;
// 文件名与目标路径冲突时 解决策略
info.conflictStrategy = QCloudSMHConflictStrategyEnumRename;
req.batchInfos = @[info];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError * _Nullable
error) {
```

```
    //如果是同步任务，从 http 的状态码中获取任务的状态
    QCloudSMHBatchTaskStatus status =
QCloudSMHBatchTaskStatusTypeFromStatus([result
__originHTTPURLResponse__].statusCode);
    result.status = status;
    if(status != QCloudSMHBatchTaskStatusWating || error){
        // 当任务状态非等待或者 有 error 时 结束
        if(self.finishBlock){
            self.finishBlock(result, error);
        }
    }else{
        // 使用返回的 taskid 进行轮询任务状态
        result.taskId;
    }

}];
[[QCloudSMHService defaultSMHService]batchMove:req];
```

查询任务状态

这里需要进行轮询任务状态，直到查询到任务结果。

```
QCloudGetTaskStatusRequest *req = [QCloudGetTaskStatusRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
// 上一步返回的 taskId
req.taskIdList = @[taskId];
[req setFinishBlock:^(NSArray * _Nonnull result, NSError * _Nonnull
error) {

}];
[[QCloudSMHService defaultSMHService] getTaskStatus:req];
```

高级批量移动示例代码

该接口封装了批量移动以及轮询任务状态，setFinishBlock 直接返回最终任务结果，无需手动查询任务状态。

```
QCloudSMHMoveObjectRequest *req = [QCloudSMHMoveObjectRequest new];
req.spaceId = @"spaceId";
req.libraryId = @"libraryId";
QCloudSMHBatchMoveInfo *info = [QCloudSMHBatchMoveInfo new];
// 原路径
```

```
info.from = @"from";
// 目标路径
info.to = @"target";
// 是否移动权限
info.moveAuthority = YES;
// 文件名与目标路径冲突时 解决策略
info.conflictStrategy = QCloudSMHConflictStrategyEnumRename;

req.batchInfos = @[info];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] moveObject:req];
```

文件
删除单个文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于删除文件的 API 概览以及 SDK 示例代码。

API	操作描述
删除文件	删除文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

删除文件

功能说明

用于删除文件。

示例代码

```
QCloudSMHDeleteFileRequest * request = [QCloudSMHDeleteFileRequest new];
request.libraryId = @"libraryId";
request.spaceId = @"spaceId";
request.filePath = @"test.jpg";
// 当媒体库开启回收站时，则该参数指定将文件移入回收站还是永久删除文件，1：永久删除，
0：移入回收站，默认为 0
request.permanent = 0;
[request setFinishBlock:^(id  _Nullable outputObject, NSError *
_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] deleteFile:request];
```

复制单个文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
重命名文件	重命名文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

重命名文件

功能说明

用于重命名文件。

示例代码

```
QCloudSMHRenameFileRequest *req = [QCloudSMHRenameFileRequest new];
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 目标文件名
req.from = @"targetname";
// 完整文件路径
req.filePath = @"filePath";
/// 文件名冲突时的处理方式，默认为 rename
/// ask: 冲突时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，
/// rename: 冲突时自动重命名文件
/// overwrite: 如果冲突目标为目录时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，否则覆盖已有文件，；
req.conflictStrategy = QCloudSMHConflictStrategyEnumOverWrite;
[req setFinishBlock:^(QCloudSMHRenameResult *result, NSError *_Nullable error) {
}];
```

```
[[QCloudSMHService defaultSMHService] renameFile:req];
```

检查文件状态

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于检查文件状态的 API 概览以及 SDK 示例代码。

API	操作描述
检查文件状态	检查文件状态

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

检查文件状态

功能说明

用于检查文件状态。

示例代码

```
QCloudSMHHeadFileRequest * request = [QCloudSMHHeadFileRequest new];
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
requeset.spaceId = @"spaceId";
// 媒体库 ID，必选参数
requeset.libraryId = @"libraryId";
// 完整文件路径，例如 /api/v1/file/smhxxx/-/foo/bar/file.docx
request.filePath = @"filePath";
// 历史版本 ID，用于获取不同版本的文件内容，可选参数，不传默认为最新版；
request.historyId = @"historyId";
[request setFinishBlock:^(id  _Nullable outputObject, NSError *
_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] headFile:request];
```

移动单个文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于移动或重命名文件的 API 概览以及 SDK 示例代码。

API	操作描述
移动或重命名文件	移动或重命名文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

移动或重命名文件

功能说明

移动或重命名文件。

示例代码

```
QCloudSMHRenameFileRequest *req = [QCloudSMHRenameFileRequest new];
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 目标文件名
req.from = @"targetname";
// 完整文件路径
req.filePath = @"filePath";
/// 文件名冲突时的处理方式，默认为 rename
/// ask: 冲突时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，
/// rename: 冲突时自动重命名文件
/// overwrite: 如果冲突目标为目录时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，否则覆盖已有文件，；
req.conflictStrategy = QCloudSMHConflictStrategyEnumOverWrite;
[req setFinishBlock:^(QCloudSMHRenameResult *result, NSError *_Nullable error) {
}];
```

```
[[QCloudSMHService defaultSMHService] renameFile:req];
```

获取文件信息

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于获取文件信息的 API 概览以及 SDK 示例代码。

API	操作描述
获取文件下载链接和信息	获取文件下载链接和信息

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

获取文件信息

功能说明

用于获取文件信息。

示例代码

```
QCloudSMHGetDownloadInfoRequest * requeset =
[[QCloudSMHGetDownloadInfoRequest alloc]init];
// 空间所在组织id,仅访问外部群组时需要填写该字段;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
requeset.spaceId = @"spaceId";
// 媒体库 ID，必选参数
requeset.libraryId = @"libraryId";
// 版本id
requeset.historyId = @"historVersionId";
// 文件路径
requeset.filePath = @"filePath";
[requeset setFinishBlock:^(QCloudSMHDownloadInfoModel * outputObject,
NSError * _Nullable error) {

}];
[[QCloudSMHService defaultSMHService] getDonwloadInfo:requeset];
```

获取文件链接

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于获取文件下载链接、文档预览链接、缩略图链接的 API 概览以及 SDK 示例代码。

API	操作描述
获取文档预览	获取文档预览链接
获取缩略图	获取照片/视频封面缩略图链接

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

获取文档预览

功能说明

用于获取文档预览链接。

示例代码

```
QCloudSMHGetPresignedURLRequest *req = [QCloudSMHGetPresignedURLRequest
new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 版本id
req.historyId = @"historVersionId";
// 文件路径
req.filePath = @"filePath";
req.purpose = QCloudSMHPurposePreview;
[req setFinishBlock:^(NSString *result, NSError *_Nullable error) {
    // result 预览链接。
}];
[[QCloudSMHService defaultSMHService] getPresignedURL:req];
```

获取缩略图

功能说明

用于获取缩略图链接。

示例代码

```
QCloudSMHGetPresignedURLRequest *req = [QCloudSMHGetPresignedURLRequest
new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 版本id
req.historyId = @"historVersionId";
// 文件路径
req.filePath = @"filePath";
// 帧数，针对 gif 的降帧处理；
req.frameNumber = 10;
// size  scale  heightSize  widthSize 等图片变换参数请查看api文档
req.size = size;
req.purpose = QCloudSMHPurposePreview;
[req setFinishBlock:^(NSString *result, NSError *_Nullable error) {
    // result 预览链接。
}];
[[QCloudSMHService defaultSMHService] getPresignedURL:req];
```

重命名文件

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于移动或重命名文件的 API 概览以及 SDK 示例代码。

API	操作描述
移动或重命名文件	移动或重命名文件

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

移动或重命名文件

功能说明

用于移动或重命名文件。

示例代码

```
QCloudSMHRenameFileRequest *req = [QCloudSMHRenameFileRequest new];
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 目标文件名
req.from = @"targetname";
// 完整文件路径
req.filePath = @"filePath";
/// 文件名冲突时的处理方式，默认为 rename
/// ask: 冲突时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，
/// rename: 冲突时自动重命名文件
/// overwrite: 如果冲突目标为目录时返回 HTTP 409 Conflict 及 SameNameDirectoryOrFileExists 错误码，否则覆盖已有文件，；
req.conflictStrategy = QCloudSMHConflictStrategyEnumOverWrite;
[req setFinishBlock:^(QCloudSMHRenameResult *result, NSError *_Nullable error) {
}];
```

```
[[QCloudSMHService defaultSMHService] renameFile:req];
```

搜索

最近更新时间：2024-10-15 20:05:41

简介

本文档提供关于搜索文件/目录的 API 概览以及 SDK 示例代码。

API	操作描述
搜索目录与文件	用于搜索目录与文件
继续获取搜索结果	用于继续获取搜索结果
删除搜索任务	用于删除搜索任务

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

搜索目录与文件

功能说明

初始化搜索，可能会返回一定量的搜索结果。

示例代码

```
QCloudSMHInitiateSearchRequest *req = [QCloudSMHInitiateSearchRequest
new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId"
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 搜索范围，指定搜索的目录，如搜索根目录可指定为空字符串、“/”或不指定该字段
req.scope = scopeDir.totalPath;
// 搜索关键字，可使用空格分隔多个关键字，关键字之间为“或”的关系并优先展示匹配关键字较
多的项目；
req.keyword = @"keyWord";
// 搜索类型，字符串或字符串数组 QCloudSMHSearchType
req.searchTypes = @[@(type)];
[req setFinishBlock:^(QCloudSMHSearchListInfo *_Nullable result, NSError
*_Nullable error) {
```

```
}];
[[QCloudSMHService defaultSMHService] initSearch:req];
```

继续获取搜索结果

功能说明

继续获取搜索结果。

示例代码

```
QCloudSMHResumeSearchRequest *req = [QCloudSMHResumeSearchRequest new];
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
// 搜索任务 ID 初始化搜索时返回
req.searchId = searchId;
// 分页标识，创建搜索任务时或继续获取搜索结果时返回的 nextMarker 字段；
req.nextMarker = nextMarker;
[req setFinishBlock:^(QCloudSMHSearchListInfo *_Nullable result, NSError
*_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] resumeSearch:req];
```

删除搜索

功能说明

用于删除搜索任务。

示例代码

```
QCloudSMHAbortSearchRequest *req = [QCloudSMHAbortSearchRequest new];
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
// 搜索任务 ID
```

```
req.searchId = @"searchId";
[req setFinishBlock:^(QCloudSMHSearchListInfo *_Nullable result, NSError
*_Nullable error) {
    if (error) {
        [liveData postError:error];
        return;
    } else {
        [liveData postValueAndComplete:searchId];
    }
}];
[[QCloudSMHService defaultSMHService] abortSearch:req];
```

目录或相簿
共享文件夹

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于获取我共享的文件夹的 API 概览以及 SDK 示例代码。

API	操作描述
获取我共享的文件夹	用于获取我共享的文件夹

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

获取我共享的文件夹

功能说明

用于获取我共享的文件夹。

示例代码

列出文件夹列表

```
QCloudSMHGetMyAuthorizedDirectoryRequest *req =
[QCloudSMHGetMyAuthorizedDirectoryRequest new];

// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;

// 分页码，默认第一页，可选参数；
req.page = 0;
// 分页大小，默认 20，可选参数；
req.pageSize = 10;
// 按名称排序为 QCloudSMHSortTypeName，
// 按修改时间排序为 QCloudSMHSortTypeMTime，
// 按文件大小排序为 QCloudSMHSortTypeSize，
// 按创建时间排序为 QCloudSMHSortTypeCTime
req.sortType = sortType;
```

```
[req setFinishBlock:^(QCloudSMHContentListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] getMyAuthorizedDirectory:req];
```

通过 marker + limit 的方式列出

```
QCloudSMHGetMyAuthorizedDirectoryRequest *req =
[QCloudSMHGetMyAuthorizedDirectoryRequest new];

// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 限制响应体中的条目数，如不指定则默认为 1000；
req.limit = limit;
// 分页标记，当需要分页时，响应体中将返回下一次请求时用于该参数的值，当请求第一页时无
需指定该参数
req.marker = marker;
// 按名称排序为 QCloudSMHSortTypeName，
// 按修改时间排序为 QCloudSMHSortTypeMTime，
// 按文件大小排序为 QCloudSMHSortTypeSize，
// 按创建时间排序为 QCloudSMHSortTypeCTime
req.sortType = sortType;
[req setFinishBlock:^(QCloudSMHContentListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] getMyAuthorizedDirectory:req];
```

列出内容

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于列出目录或相簿内容的 API 概览以及 SDK 示例代码。

API	操作描述
列出目录或相簿	用于列出目录或相簿内容

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

列出目录或相簿

功能说明

用于列出目录或相簿内容。

示例代码

列出文件列表

```
QCloudSMHListContentsRequest *req = [QCloudSMHListContentsRequest new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
req.dirPath = @"/";
req.page = 0;
req.pageSize = 10;
req.requestSerializer.cachePolicy = NSURLRequestReloadIgnoringCacheData;
[req setFinishBlock:^(QCloudSMHContentListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] listContents:req];
```

通过 marker + limit 的方式列出文件列表

```
//目标文件夹
QCloudSMHListContentsRequest *req = [QCloudSMHListContentsRequest new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";
req.dirPath = @"/";
// 限制响应体中的条目数，如不指定则默认为 1000;
req.limit = 0;
// 分页标记，当需要分页时，响应体中将返回下一次请求时用于该参数的值，当请求第一页时无需指定该参数
req.marker = @"marker";
req.requestSerializer.cachePolicy = NSURLRequestReloadIgnoringCacheData;
[req setFinishBlock:^(QCloudSMHContentListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] listContents:req];
```

创建文件夹

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
创建目录或相簿	创建目录或相簿

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

创建目录或相簿

功能说明

用于创建目录或相簿。

示例代码

```
QCloudSMHPutDirectoryRequest *req = [QCloudSMHPutDirectoryRequest new];
req.dirPath = @"dirname";
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";

[req setFinishBlock:^(QCloudSMHContentInfo *contentInfo, NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] putDirectory:req];
```

删除单个文件夹

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
删除目录或相簿	删除目录或相簿

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

删除目录或相簿

功能说明

用于删除目录或相簿。

示例代码

```
QCloudSMHDeleteObjectRequest *req = [QCloudSMHDeleteObjectRequest new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = @"spaceId";
NSMutableArray *batchInfos = [NSMutableArray array];
QCloudSMHBatchDeleteInfo *info = [QCloudSMHBatchDeleteInfo new];
// 被删除的目录、相簿或文件路径；
info.path = testDirName;
[batchInfos addObject:info];

req.batchInfos = [batchInfos copy];
[req setFinishBlock:^(QCloudSMHBatchResult *result, NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService]deleteObject:req];
```

获取相簿封面

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于查看目录或相簿的 API 概览以及 SDK 示例代码。

API	操作描述
获取相簿封面	获取相簿封面

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

获取相簿封面链接

功能说明

用于获取相簿封面链接。

示例代码

```
QCloudSMHGetAlbumRequest * request = [QCloudSMHGetAlbumRequest new];
// 缩放大小，可选参数，相关说明参阅接口说明。
request.size = @"100";
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
[request setFinishBlock:^(id  _Nullable outputObject, NSError *
_Nullable error) {
    [expectation fulfill];
}];
[[QCloudSMHService defaultSMHService] getAlbum:request];
```

重命名文件夹

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于文件重命名的 API 概览以及 SDK 示例代码。

API	操作描述
重命名或移动目录或相簿	重命名或移动目录或相簿

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

重命名或移动目录或相簿

功能说明

用于重命名或移动目录或相簿。

示例代码

```
QCloudSMHRenameDirectoryRequest *req = [QCloudSMHRenameDirectoryRequest
new];
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
// 目录路径或相簿名，对于多级目录，使用斜杠(/)分隔，例如 foo/bar
req.dirPath = name;
// 定被重命名或移动的源目录路径或相簿名
req.from = cto.object.totalPath;
req.conflictStrategy = QCloudSMHConflictStrategyEnumAsk;
// 是否移动文件夹权限，true 移动，false 不移动；
req.moveAuthority = YES;
[req setFinishBlock:^(QCloudSMHRenameResult *result, NSError *_Nullable
error) {
    if (completeHandler) {
        completeHandler(result, error);
    }
}];
```

```
[[QCloudSMHService defaultSMHService] renameDirecotry:req];
```

角色权限
获取角色列表

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于用于获取角色列表的 API 概览以及 SDK 示例代码。

API	操作描述
获取角色列表	用于获取角色列表

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

获取角色列表

功能说明

用于获取角色列表。

示例代码

```
QCloudSMHGetRoleListRequest *request = [[QCloudSMHGetRoleListRequest
alloc] init];
// 媒体库 ID，必选参数
request.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
request.spaceId = @"spaceId";
[request setFinishBlock:^(id _Nullable outputObject, NSError *_Nullable
error) {
    [liveData postError:error value:outputObject];
}];
[[QCloudSMHService defaultSMHService] getRoleList:request];
```

回收站
删除回收站文件

最近更新时间：2025-04-30 16:05:02

简介

本文档提供关于删除回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
删除回收站项目	删除回收站项目
清空回收站	清空回收站

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

删除回收站项目

功能说明

用于批量删除回收站项目。

示例代码

```
QCloudSMHDeleteRecycleObjectReqeust *req =
[QCloudSMHDeleteRecycleObjectReqeust new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 回收站项目 ID，必选参数；
req.recycledItemId = @"1";
[req setFinishBlock:^(id _Nullable outputObject, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] deleteRecycleObject:req];
```

清空回收站

功能说明

用于清空回收站。

示例代码

```
QCloudSMHDeleteAllRecycleObjectReqeust *req =
[QCloudSMHDeleteAllRecycleObjectReqeust new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";

[req setFinishBlock:^(id _Nullable outputObject, NSError *_Nullable
error) {

}];
[[QCloudSMHService defaultSMHService] deleteAllRecyleObject:req];
```

回收站列表

最近更新时间：2024-10-15 20:05:42

简介

本文档提供关于列出回收站项目的 API 概览以及 SDK 示例代码。

API	操作描述
列出回收站项目	用于列出回收站项目

SDK API 参考

SDK 所有接口的具体参数与方法说明，请参见 SDK API 参考 。

列出回收站项目

功能说明

用于列出回收站项目。

示例代码

列出回收站列表

```
QCloudSMHGetRecycleObjectListReqeust *req =
[QCloudSMHGetRecycleObjectListReqeust new];
// 媒体库 ID，必选参数
req.libraryId = @"libraryId";
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，
则必须指定该参数
req.spaceId = @"spaceId";
// 分页码，默认第一页，可选参数；
req.page = 0;
// 分页大小，默认 20，可选参数；
req.pageSize = 10;
// 按名称排序为 QCloudSMHSortTypeName，
// 按修改时间排序为 QCloudSMHSortTypeMTime，
// 按文件大小排序为 QCloudSMHSortTypeSize，
// 按删除时间排序为 QCloudSMHSortTypeRemovalTime，
// 按剩余时间排序为 QCloudSMHSortTypeRemainingTime；
req.sortType = sortType;
// 分页方向，当请求下一页时传 next，当请求上一页时，传 prev；
```

```
req.isNext = YES;
[req setFinishBlock:^(QCloudSMHRecycleObjectListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] getRecycleList:req];
```

通过 marker + limit 的方式列出

```
QCloudSMHGetRecycleObjectListReqeust *req =
[QCloudSMHGetRecycleObjectListReqeust new];
// 媒体库 ID，必选参数
req.libraryId = self.userModel.libraryId;
// 空间 ID，如果媒体库为单租户模式，则该参数固定为连字符(-)；如果媒体库为多租户模式，则必须指定该参数
req.spaceId = scopeDir.teamInfo.spaceId;
// 用户身份识别，当访问令牌对应的权限为管理员权限且申请访问令牌时的用户身份识别为空时用来临时指定用户身份，详情请参阅生成访问令牌接口
req.userId = self.userModel.userId;
// 限制响应体中的条目数，如不指定则默认为 1000；
req.limit = limit;
// 分页标记，当需要分页时，响应体中将返回下一次请求时用于该参数的值，当请求第一页时无需指定该参数
req.marker = marker;
// 排序字段，
// 按名称排序为 QCloudSMHSortTypeName，
// 按修改时间排序为 QCloudSMHSortTypeMTime，
// 按文件大小排序为 QCloudSMHSortTypeSize，
// 按删除时间排序为 QCloudSMHSortTypeRemovalTime，
// 按剩余时间排序为 QCloudSMHSortTypeRemainingTime；
req.sortType = sortType;
// 分页方向，当请求下一页时传 next，当请求上一页时，传 prev；
req.isNext = YES;
[req setFinishBlock:^(QCloudSMHRecycleObjectListInfo *_Nullable result,
NSError *_Nullable error) {

}];
[[QCloudSMHService defaultSMHService] getRecycleList:req];
```