目录

SDK概览

金山云存储为了更好的服务广大用户，帮助用户更方便的使用我们的服务，我们提供了丰富的SDK，这些SDK涵盖了各种语言及平台，目前主要包括：

- Java
- Go
- Python
- Node.js
- JavaScript
- PHP
- iOS
- C#
- C/C++
- Android

安装

注意

文档中的示例代码仅供参考，具体使用的时候请参考KS3 API文档，根据自己的实际情况调节参数。

lib目录下为该项目所依赖的所有jar包，以及将sdk打好的jar包。

概述

此SDK适用于Java 7及以上版本。基于KS3 API 构建。

环境准备

- 配置Java 7 以上开发环境
- 下载 KS3 SDK For Java
- 添加Maven依赖

```
<dependency>
    <groupId>com.ksyun</groupId>
    <artifactId>ks3-kss-java-sdk</artifactId>
    <version>1.0.4</version>
</dependency>
```

- 或者直接引用lib目录下的所有jar包

初始化

Ks3Client 是KS3的Java客户端，用于管理存储空间和文件等KS3资源。使用Java SDK发起KS3请求，您需要初始化一个Ks3Client实例，并根据需要修改Ks3ClientConfig的默认配置项。

获取秘钥

1. 开通KS3服务，注册账号
2. 进入控制台，获取AccessKeyID 、AccessKeySecret

新建Ks3Client

新建Ks3Client时，需要指定Endpoint。有关Endpoint的更多信息，请参见访问域名和数据中心和自定义访问域名。

以下代码用于新建Ks3Client。一个工程中可以有一个或多个Ks3Client，Ks3Client支持并发使用。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
```

```
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
 * 设置domainMode
 * true：表示以自定义域名访问
 * false：表示以KS3的外网域名或内网域名访问，默认为false
*/
config.setDomainMode(false);
// 设置通信协议，可选项 Ks3ClientConfig.PROTOCOL.http 和 Ks3ClientConfig.PROTOCOL.https
config.setProtocol(Ks3ClientConfig.PROTOCOL.http);
/**
 * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);
// 设置httpclient
HttpClientConfig hconfig = new HttpClientConfig();
// 在HttpClientConfig中可以设置httpclient的相关属性，比如代理，超时，重试等。
config.setHttpClientConfig(hconfig);
// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);
```

配置日志

SDK使用commons-logging

- 使用log4j的示例：

1. 引用log4j相关jar包

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
```

2. 新建log4j.properties(如下为示例配置)

```
log4j.logger.com.ksyun.ks3=DEBUG,stdout
log4j.logger.org.apache.http=DEBUG,stdout
log4j.logger.org.apache.http.wire=ERROR,stdout
log4j.addivity.org.apache=true
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS} %-5p [%t]%C{1}.%M(%L) - %m%n
```

- 使用logback示例

引用SDK的时候排除commons-logging，引用logback相关包（包括但不止jcl-over-slf4j）。

```
<dependency>
    <groupId>com.ksyun</groupId>
    <artifactId>ks3-kss-java-sdk</artifactId>
    <version>0.6.0</version>
    <exclusions>
        <exclusion>
            <artifactId>commons-logging</artifactId>
            <groupId>commons-logging</groupId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.7</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.3</version>
</dependency>
```

常见术语介绍

Object（对象，文件）

在KS3中，用户操作的基本数据单元是Object。单个Object允许存储0~48.8TB的数据。 Object 包含key和data。其中，key是Object的名字；data是Object 的数据。key为UTF-8编码，且编码后的长度不得超过1024个字符。

Key（文件名）

即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字符。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

其他术语请参考概念与术语

快速入门

请先阅读常用概念术语文档。

新建一个存储空间

存储空间（Bucket）是存储对象（Object）的容器。对象都隶属于存储空间。本文介绍如何创建存储空间。

以下代码用于创建examplebucket存储空间。

您可以在创建存储空间时指定存储类型和存储空间读写权限。更多信息，请分别参见存储类型介绍和设置存储空间读写权限（ACL）。

控制台创建

进入控制台，点击右上角新建空间，新建的空间名称即为之后提到的bucket。

SDK 创建

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
 * false:  （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);
// 创建CreateBucketRequest对象。
CreateBucketRequest createBucketRequest = new CreateBucketRequest("examplebucket");
// 如果创建存储空间的同时需要指定存储类型和数据容灾类型, 请参考如下代码。
// 此处以设置存储空间的存储类型为标准存储(默认是标准存储)为例介绍。
//createBucketRequest.setBucketType(BucketType.Normal);
// 设置存储空间的权限为公共读写，默认为私有。
//createBucketRequest.setCannedAcl(CannedAccessControlList.PublicReadWrite);
client.createBucket(createBucketRequest);
```

上传文件

简单上传是指通过PutObject方法上传单个文件（Object）。简单上传包括流式上传和文件上传，流式上传使用InputStream作为KS3文件的数据源，文件上传使用本地文件作为KS3文件的数据源。本文介绍如何使用流式上传和文件上传方式上传文件。

流式上传

- 上传字符串

  以下代码用于将字符串上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
```

```
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.consol
e.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
 * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);

// 填写字符串。
String content = "Hello KS3";
// 设置上传文件的meta信息
ObjectMetadata metadata = new ObjectMetadata();
// 创建PutObjectRequest对象。
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
PutObjectRequest putObjectRequest = new PutObjectRequest("examplebucket", "exampledir/exampleobject.txt", new ByteArrayInputStream
(content.getBytes()), metadata);

// 如果需要上传时设置存储类型和访问权限，请参考以下示例代码。
// putObjectRequest.setStorageClass(StorageClass.Standard);
// putObjectRequest.setCannedAcl(CannedAccessControlList.Private);

// 上传字符串。
client.putObject(putObjectRequest);
```

- 上传Byte数组

  以下代码用于将Byte数组上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设
置endpoint为自定义域名，同时设置domainMode为true
  String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.consol
e.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
* false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
* true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
* 如果domainMode设置为true，则pathStyleAccess可忽略设置
*/
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);

// 填写Byte数组。
byte[] content = "Hello KS3".getBytes();
// 设置上传文件的meta信息
ObjectMetadata metadata = new ObjectMetadata();
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
client.putObject("examplebucket", "exampledir/exampleobject.txt", new ByteArrayInputStream(content), metadata);
```

- 上传网络流

  以下代码用于将网络流上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设
置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.consol
e.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
```

```
 * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);

// 填写网络流地址。
InputStream inputStream = new URL("https://www.ksyun.com").openStream();
// 设置上传文件的meta信息
  ObjectMetadata metadata = new ObjectMetadata();
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
client.putObject("examplebucket", "exampledir/exampleobject.txt", inputStream, metadata);
```

文件上传

以下代码用于将文件流上传到目标存储空间examplebucket中exampledir目录下的exampleobject.txt文件。

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
 * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId,accessKeySecret,config);
// 填写本地文件的完整路径。如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件流。
File file = new File("D:\\localpath\\examplefile.txt");
// 依次填写Bucket名称（例如examplebucket）和Object完整路径（例如exampledir/exampleobject.txt）。Object完整路径中不能包含Bucket名称。
client.putObject("examplebucket", "exampledir/exampleobject.txt", file);
```

通过表单的方式上传文件

1. 了解KS3表单上传协议，以及表单上传签名认证方式。

2. 在KS3控制台->空间设置->CORS配置里配置CORS跨域规则。CORS为跨域资源共享，当使用js跨域时，需要配置该规则。W3C文档。

3. 使用js sdk上传文件: 。

4. js sdk 中的policy和signature可以通过Java SDK的该方法计算出来。

```
public PostObjectFormFields postObjectSimple() {
        // yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
        String endpoint = "yourEndpoint";
        // 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
        String accessKeyId = "yourAccessKeyId";
        String accessKeySecret = "yourAccessKeySecret";
        // 创建Ks3ClientConfig 实例。
        Ks3ClientConfig config = new Ks3ClientConfig();
        // 设置域名
        config.setEndpoint(endpoint);
        /**
         * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
         * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
         * 如果domainMode设置为true，则pathStyleAccess可忽略设置
         */
        config.setPathStyleAccess(false);

        // 创建Ks3Client实例
        Ks3 client = new Ks3Client(accessKeyId, accessKeySecret, config);
        /**
        * 需要用户在postData和unknowValueField中提供所有的除KSSAccessKeyId, signature, file, policy外的所有表单项。否则用生成的签名上传会返回403</br>
```

```
        * 对于用户可以确定表单值的放在 postData中，对于用户无法确定表单值的放在unknownValueField中(比如有的上传控件会添加一些表单项,但表单项的值可能是随机的)</br>
        */
        Map<String, String> postData = new HashMap<String, String>();
        // 如果使用js sdk上传的时候设置了ACL，请提供以下一行，且值要与SDK中一致，否则删除下面一行代码
        postData.put("acl","public-read");
        // 提供js sdk中的key值
        postData.put("key","20150115/中文/${filename}");
        // 设置无法确定的表单值
        List<String> unknowValueField = new ArrayList<String>();
        // js sdk上传的时候会自动加上一个name的表单项，所以下面需要加上这样的代码。
        unknowValueField.add("name");
        // 如果计算签名时提供的key里不包含${filename}占位符，第二个参数传一个空字符串。
        PostObjectFormFields fields = client.postObject("<您的bucket名称>", "<要上传的文件名称,不包含路径信息>", postData, unknowValueField);
        // 获取KssAccessKeyId
        fields.getKssAccessKeyId();
        // 获取 post 的 policy
        fields.getPolicy();
        // 获取 signature
        fields.getSignature();
        // 返回 policy 相关信息
        return fields;
}
```

常见问题：

- 上传时，浏览器一共会发送两个请求，首先是OPTIONS请求，如果该请求返回403，请检测CORS配置是否正确。第二个为POST请求，如果该请求返回403，请检查生成policy时，是否按照表单内容生成，或者把policy进行base64解码，对比表单内容是否正确。对比规则：Policy、Signature构建方法。

获取文件访问地址

- 如果是公开文件 通过：http://{bucket}.{endpoint}/{key}的方式拼接一个URL即可。比如：http://test-bucket.ks3-cn-beijing.ksyuncs.com/2015/10/19/image.jpg， 该URL中的{bucket}是test-bucket,{endpoint}是kssws.ks-cdn.com,{key}是2015/10/19/image.jpg。
- 如果是私有文件 通过以下代码可以生成一个访问地址

```
// yourEndpoint填写Bucket所在地域对应的Endpoint。以中国（北京）为例，
Endpoint填写为ks3-cn-beijing.ksyuncs.com。如果使用自定义域名，设置endpoint为自定义域名，同时设置domainMode为true
String endpoint = "yourEndpoint";
// 金山云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录
https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
String accessKeyId = "yourAccessKeyId";
String accessKeySecret = "yourAccessKeySecret";
// 创建Ks3ClientConfig 实例。
Ks3ClientConfig config = new Ks3ClientConfig();
// 设置域名
config.setEndpoint(endpoint);
/**
 * false: （推荐）使用三级域名：{bucketName}.{endpoint}/{objectKey}的形式访问
 * true:  使用二级域名：{endpoint}/{bucketName}/{objectKey}的形式访问
 * 如果domainMode设置为true，则pathStyleAccess可忽略设置
 */
config.setPathStyleAccess(false);

// 创建Ks3Client实例
Ks3 client = new Ks3Client(accessKeyId, accessKeySecret, config);
// 生成一个在1000秒后过期的外链
client.generatePresignedUrl(<bucket>,<key>,1000);
// 生成一个1000秒后过期并重写返回的heade的外链
ResponseHeaderOverrides overrides = new ResponseHeaderOverrides();
// overrides.setContentType("text/html");
// .......
// 生成指定 bucket 和 key 的外链，有效期1000秒
String url = client.generatePresignedUrl(<bucket>,<key>,1000,overrides);
```

常见问题：

- 如果文件不存在，会返回NoSuchKey错误。
- 如果以公开的方式访问私有文件，会返回AccessDined错误。
- 如果私有文件访问地址过期，会返回URLExpired错误。
- 1000秒后过期是参照客户端本地时间的。

存储空间

上传文件

下载文件

管理文件

对象标签

数据加密

图片处理

1. 根据图片处理API和图片样式管理在原有key的基础上拼接图片处理参数。比如：{原key}@base@tag=imageInfo 或者 {原key}@style@{样式名称}。

2. 使用拼接好的key，获取文件访问地址即可。

安装

本文介绍如何安装Go SDK

环境准备

环境要求

- 使用Golang 1.6及以上版本。
- 请参考Golang 官方网站下载和安装Go编译运行环境。
- 安装完成后，请新建系统变量GOPATH，并将其指向您的代码目录。
- 要了解更多GOPATH的信息，请执行以下命令：

  ```
  go help gopath
  ```

  #### 查看语言版本

  要查看Go语言版本，请执行以下命令：

  ```
  go version
  ```

  ### 安装方式

使用以下命令安装Go SDK：

```
go get github.com/ks3sdklib/aws-sdk-go
```

SDK下载和示例

- SDK 下载，请访问 Github/SDK。
- 使用示例，请参考 Github/test。

请参考上述链接获取SDK和示例代码。 如果您需要了解更多关于Go语言版本和GOPATH相关信息，可以使用以下命令：

- 查看Go语言版本：`go version`
- 获取GOPATH相关信息：`go help gopath`

初始化

以下代码用于初始化客户端：

```
// 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
// 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录https://uc.console.ksyun.com/pro/iam/#/user/list创建子账号。
// 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
  credentials := credentials.NewStaticCredentials("<AccessKeyID>","<AccessKeySecret>","")
    client = s3.New(&aws.Config{
        //Region 可参考 https://docs.ksyun.com/documents/6761
        Region:      region,
        Credentials: cre,
        //Endpoint 可参考 https://docs.ksyun.com/documents/6761
        Endpoint:          endpoint,
        DisableSSL:        true,  //是否禁用https
        LogLevel:          0,     //是否开启日志,0为关闭日志，1为开启日志
        LogHTTPBody:       false, //是否把HTTP请求body打入日志
        S3ForcePathStyle: false,
        Logger:            nil,   //打日志的位置
        DomainMode:        false, //是否开启自定义bucket绑定域名，当开启时 S3ForcePathStyle 参数不生效。
        //可选值有 ： V2 OR V4 OR V4_UNSIGNED_PAYLOAD_SIGNER
        SignerVersion: "V4",
        MaxRetries:    1,
    })
```

注意：

- endpoint 与 Region 对应关系
- 关于获取AK/SK的更多细节，请参见文档：开通KS3服务

快速入门

创建存储空间

以下代码用于创建存储空间：

```
package bucket_sample

import (
    "fmt"
    "github.com/ks3sdklib/aws-sdk-go/aws"
    "github.com/ks3sdklib/aws-sdk-go/aws/awsutil"
    "github.com/ks3sdklib/aws-sdk-go/aws/credentials"
    "github.com/ks3sdklib/aws-sdk-go/service/s3"
)

var bucket = string("yourbucket")
var key = string("yourkey")
var key_encode = string("yourkey")
var key_copy = string("yourkey")
var content = string("content")

// 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
// 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
// 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
var cre = credentials.NewStaticCredentials("ak", "sk", "") //online
var svc = s3.New(&aws.Config{
        Region:      "BEIJING",
    Credentials: cre,
    //Endpoint:"ks3-sgp.ksyun.com",
    Endpoint:          "ks3-cn-beijing.ksyuncs.com",
    DisableSSL:        true,
    LogLevel:          1,
    S3ForcePathStyle: false,
    LogHTTPBody:       true,
})

//创建bucket并关联项目
func TestCreateBucket(svc *s3.S3) {
        resp, err := svc.CreateBucket(&s3.CreateBucketInput{
            ACL:    aws.String("public-read"),//权限
            Bucket: aws.String("bucket"),
            ProjectId: aws.String("123123"),//项目制id
        })
        if err != nil {
            if awsErr, ok := err.(awserr.Error); ok {
                fmt.Println(awsErr.Code(), awsErr.Message(), awsErr.OrigErr())
                if reqErr, ok := err.(awserr.RequestFailure); ok {
                    fmt.Println(reqErr.Code(), reqErr.Message(), reqErr.StatusCode(), reqErr.RequestID())
                }
            } else {
```

```
                fmt.Println(err.Error())
            }
        }
        fmt.Println("结果：\n", awsutil.StringValue(resp))
}
```

- 存储空间的命名规范请参见空间管理-基本操作。
- 您可以在创建存储空间时指定存储空间的权限和存储类型。

上传对象

以下代码用于上传文件：

```
package ks3test

import (
    "bufio"
    "bytes"
    "github.com/ks3sdklib/aws-sdk-go/aws/awserr"
    "github.com/ks3sdklib/aws-sdk-go/aws/awsutil"
    "net/url"
    "strconv"
    "strings"
    "testing"
    "time"
    //  "io"
    "fmt"
    "github.com/ks3sdklib/aws-sdk-go/aws"
    "github.com/ks3sdklib/aws-sdk-go/aws/credentials"
    "github.com/ks3sdklib/aws-sdk-go/service/s3"
    "net/http"
    "os"
)

var bucket = string("yourbucket")
var key = string("yourkey")
var key_encode = string("yourkey")
var key_copy = string("yourkey")
var content = string("content")
var prefix = "test/" //目录名称

// 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
// 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
// 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
var cre = credentials.NewStaticCredentials("ak", "sk", "") //online
var svc = s3.New(&aws.Config{
    //Region 可参考 [https://docs.ksyun.com/documents/6761](https://docs.ksyun.com/documents/6761)
    Region:      "BEIJING",
    Credentials: cre,
    //Endpoint 可参考 [https://docs.ksyun.com/documents/6761](https://docs.ksyun.com/documents/6761)
    Endpoint:         "ks3-cn-beijing.ksyuncs.com",
    DisableSSL:       true, //是否禁用https
    LogLevel:         1,    //是否开启日志,0为关闭日志，1为开启日志
    LogHTTPBody:      true, //是否把HTTP请求body打入日志
    S3ForcePathStyle: false,
    Logger:           nil, //打日志的位置
})

//上传文件
func putObject(svc *s3.S3) {

    //获取本地文件FD
    fd, _ := os.Open("D:\\suiyi.jpg")

    //指定目标Object对象标签，可同时设置多个标签，如：TagA=A&TagB=B。
    //说明 Key和Value需要先进行URL编码，如果某项没有“=”，则看作Value为空字符串。详情请见对象标签（[https://docs.ksyun.com/documents
/39576](https://docs.ksyun.com/documents/39576)）。
    v := url.Values{}
    v.Add("name", "yz")
    v.Add("age", "11")
    XAmzTagging := v.Encode()

    input := s3.PutObjectInput{
        Bucket:      aws.String(bucket),
        Key:         aws.String(key),
        ACL:         aws.String("public-read"),
        Body:        fd,
        XAmzTagging: aws.String(XAmzTagging),
    }
    resp, _ := svc.PutObject(&input)
    fmt.Println("结果：\n", awsutil.StringValue(resp))
}
```

上传文件的更多信息，请参见 PUT Object 。

列举对象

以下代码用于列举文件：

```
package ks3test

import ( "bufio" "bytes" "github.com/ks3sdklib/aws-sdk-go/aws/awserr" "github.com/ks3sdklib/aws-sdk-go/aws/awsutil" "net/url" "strconv" "strings" "testing" "time" // "io" "fmt" "github.com/ks3sdklib/aws-sdk-go/aws" "github.com/ks3sdklib/aws-sdk-go/aws/credentials" "github.com/ks3sdklib/aws-sdk-go/service/s3" "net/http" "os" )

var bucket = string("yourbucket") var key = string("yourkey") var key_encode = string("yourkey") var key_copy = string("yourkey") var content = string("content") var prefix = "test/" //目录名称

// 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。 // 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。 // 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。 var cre = credentials.NewStaticCredentials("ak", "sk", "") //online var svc = s3.New(&aws.Config{ //Region 可参考 https://docs.ksyun.com/documents/6761 Region: "BEIJING", Credentials: cre, //Endpoint 可参考 https://docs.ksyun.com/documents/6761 Endpoint: "ks3-cn-beijing.ksyuncs.com", DisableSSL: true, //是否禁用https LogLevel: 1, //是否开启日志,0为关闭日志，1为开启日志 LogHTTPBody: true, //是否把HTTP请求body打入日志 S3ForcePathStyle: false, Logger: nil, //打日志的位置 })

//列表bucket下对象 func TestListObjects(t *testing.T) {

objects1, _ := svc.ListObjects(&s3.ListObjectsInput{
    Bucket:    aws.String(bucket),
    Delimiter: aws.String("/"),     //分隔符，用于对一组参数进行分割的字符
    MaxKeys:   aws.Long(int64(30)), //设置响应体中返回的最大记录数（最后实际返回可能小于该值）。默认为1000。如果你想要的结果在1000条以后，你可以设定 marker 的值来调整起始位置。
    Prefix:    aws.String(prefix),  //限定响应结果列表使用的前缀，正如你在电脑中使用的文件夹一样。
    Marker:    aws.String(""),      //指定列举指定空间中对象的起始位置。KS3按照字母排序方式返回结果，将从给定的 marker 开始返回列表。
})
//获取对象列表
objectList := objects1.Contents
for i := 0; i < len(objectList); i++ {
    object := objectList[i]
    println(*object.Key)
}

}

>列举文件的更多详情，请参见 [GetBucket (ListObjects)](https://docs.ksyun.com/documents/42288) 。
### 删除对象
以下代码用于删除文件

package ks3test

import ( "bufio" "bytes" "github.com/ks3sdklib/aws-sdk-go/aws/awserr" "github.com/ks3sdklib/aws-sdk-go/aws/awsutil" "net/url" "strconv" "strings" "testing" "time" // "io" "fmt" "github.com/ks3sdklib/aws-sdk-go/aws" "github.com/ks3sdklib/aws-sdk-go/aws/credentials" "github.com/ks3sdklib/aws-sdk-go/service/s3" "net/http" "os" )

var bucket = string("yourbucket") var key = string("yourkey") var key_encode = string("yourkey") var key_copy = string("yourkey") var content = string("content") var prefix = "test/" //目录名称

// 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。 // 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。 // 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。 var cre = credentials.NewStaticCredentials("ak", "sk", "") //online var svc = s3.New(&aws.Config{ //Region 可参考 https://docs.ksyun.com/documents/6761 Region: "BEIJING", Credentials: cre, //Endpoint 可参考 https://docs.ksyun.com/documents/6761 Endpoint: "ks3-cn-beijing.ksyuncs.com", DisableSSL: true, //是否禁用https LogLevel: 1, //是否开启日志,0为关闭日志，1为开启日志 LogHTTPBody: true, //是否把HTTP请求body打入日志 S3ForcePathStyle: false, Logger: nil, //打日志的位置 })

//删除对象 func TestDelObject(svc *s3.S3) {

resp, _ := svc.DeleteObject(&s3.DeleteObjectInput{
    Bucket: aws.String(bucket),
    Key:    aws.String(key),
})
fmt.Println("结果：\n", awsutil.StringValue(resp))
```

}

>删除文件的更多详情，请参见 [DELETE Object](https://docs.ksyun.com/documents/42318) 。

存储空间

上传文件

下载文件

管理文件

对象标签

安装

环境准备

适用于Python2.6、2.7、3.3、3.4、3.5、3.6、3.7版本。

开发前准备

安装依赖模块

```
pip install six
```

安装Python SDK

可以通过在线或者本地两种方式安装python sdk

在线安装

```
pip install ks3sdk
```

本地安装

通过git下载SDK到本地

```
git clone https://gitee.com/ks3sdk/ks3-python-sdk.git
```

进入ks3-python-sdk目录

```
cd  ks3-python-sdk
```

安装SDK

```
python setup.py install
```

初始化

以下代码用于初始化connection。

```
# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
from ks3.connection import Connection
ak = 'YOUR_ACCESS_KEY'
sk = 'YOUR_SECRET_KEY'

# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。host(Endpoint) 以北京为例，其它 Region 请按实际情况填写。
c = Connection(ak, sk, host='YOUR_REGION_ENDPOINT', is_secure=False, domain_mode=False)
```

参数说明

- ak：金山云提供的ACCESS KEY ID
- sk：金山云提供的SECRET KEY ID
- host：金山云提供的各个Region的域名（例 ks3-cn-beijing.ksyuncs.com），具体定义可参考 API接口文档-Region(区域)。也可以是用户自定义域名，如果是用户自定义域名，需要将domain_mode设置为True。
- is_secure：是否通过HTTPS协议访问Ks3，True:启用 False:关闭。
- domain_mode：是否使用自定义域名访问Ks3（host填写自定义域名），True:是 False:否。
- ua_addon：User-Agent的追加设置，推荐格式为{product}/{version}({comment})，比如test/1.0.0。User-Agent默认值为PythonSDK/{version}，设置后的值为

  PythonSDK/{version_id} {追加值}

- timeout：连接超时，默认10s

快速入门

用户常用操作，创建存储空间，下载文件，上传文件等。

创建存储空间

在建立了连接后，可以创建一个 Bucket。Bucket 在 ks3 中是一个用于储存 key/value 的容器。用户可以将所有的数据存储在一个 Bucket 里，也可以为不同种类数据创建相应的 Bucket。 以下代码用于创建存储空间：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

# 这里如果出现409 conflict错误，说明请求的bucket name有冲突，因为bucket name是全局唯一的。
# 默认创建私有访问权限的存储空间。
b = c.create_bucket('<YOUR_BUCKET_NAME>')

# 如果需要在创建存储空间时设置存储空间访问权限，请参考以下代码。
# 以下以配置存储空间为私有访问权限。
b = c.create_bucket('<YOUR_BUCKET_NAME>', policy='private')
```

列举存储空间

以下代码用于列举存储空间：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

#获取客户所有的bucket信息。
buckets = c.get_all_buckets()

#打印出bucket的名称。
for b in buckets:
    print(b.name)
```

下载文件

以下代码用于下载文件到本地：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

# 获取文件名称。
```

```
key_name = "YOUR_KEY_NAME"

# 获取存储空间实例。
b = c.get_bucket('<YOUR_BUCKET_NAME>')

# 获取文件实例。
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
k = b.get_key(key_name)

#下载object，并保存到该文件路径中。
k.get_contents_to_filename("<SAVED_FILE_PATH>")
```

上传文件

以下代码用于上传文件到KS3：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

# 获取存储空间实例。
b = c.get_bucket("<YOUR_BUCKET_NAME>")

# 创建文件。
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
k = b.new_key("<YOUR_KEY_NAME>")

# 将指定目录下的某一文件上传。
# 填写本地文件的完整路径。
ret=k.set_contents_from_filename("<YOUR_SOURCE_FILE_PATH>")

# HTTP返回码。
if ret and ret.status == 200:
    print("上传成功")
```

删除文件

以下代码用于删除指定文件：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

# 获取存储空间实例。
b=c.get_bucket("<YOUR_BUCKET_NAME>")

# 删除文件。<YOUR_KEY_NAME>表示删除OSS文件时需要指定包含文件后缀在内的完整路径。如 images/test.jpg。
# 暂不支持删除文件夹。
b.delete_key("<YOUR_KEY_NAME>")
```

列举文件

以下代码用于列举指定存储空间下的文件：

```
from ks3.connection import Connection
from ks3.prefix import Prefix
from ks3.key import Key

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。
c = Connection('<YOUR_ACCESS_KEY>', '<YOUR_SECRET_KEY>', host='YOUR_REGION_ENDPOINT')

#获取存储空间实例。
b = c.get_bucket("<YOUR_BUCKET_NAME>")

#列举文件信息并打印。
keys = b.list()
for k in keys:
  print('object:', k)
```

存储空间

上传文件

下载文件

管理文件

对象标签（Python）

设置对象标签

以下代码用于设置对象标签：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。host(Endpoint) 以北京为例，其它 Region 请按实际情况填写。
conn = Connection('<yourAccessKeyId>', '<yourAccessKeySecret>', host='ks3-cn-beijing.ksyuncs.com')

# 获取存储空间实例
b = conn.get_bucket('<yourBucketName>')

from ks3.objectTagging import Tag
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
k = b.get_key('<yourKeyName>')
tagging = [Tag('<key>', '<value>')]
# 设置对象标签
k.set_object_tagging(tagging)
```

- 对象标签更多信息，请参见对象标签。
- 设置对象标签的更多详情，请参见 Put Object Tagging。

获取对象标签

以下代码用于获取对象标签：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。host(Endpoint) 以北京为例，其它 Region 请按实际情况填写。
conn = Connection('<yourAccessKeyId>', '<yourAccessKeySecret>', host='ks3-cn-beijing.ksyuncs.com')

# 获取存储空间实例
b = conn.get_bucket('<yourBucketName>')

from ks3.objectTagging import Tag
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
k = b.get_key('<yourKeyName>')

# 获取对象标签
tagging = k.get_object_tagging()
print(tagging.to_xml())
```

设置对象标签的更多详情，请参见 Get Object Tagging。

删除对象标签

以下代码用于删除对象标签：

```
from ks3.connection import Connection

# 金山云主账号 AccessKey 拥有所有API的访问权限，风险很高。
# 强烈建议您创建并使用子账号账号进行 API 访问或日常运维，请登录 https://uc.console.ksyun.com/pro/iam/#/user/list 创建子账号。
# 通过指定 host(Endpoint)，您可以在指定的地域创建新的存储空间。host(Endpoint) 以北京为例，其它 Region 请按实际情况填写。
conn = Connection('<yourAccessKeyId>', '<yourAccessKeySecret>', host='ks3-cn-beijing.ksyuncs.com')

# 获取存储空间实例
b = conn.get_bucket('<yourBucketName>')
```

```
from ks3.objectTagging import Tag
# 填写Object完整路径。Object完整路径中不能包含Bucket名称。
k = b.get_key('<yourKeyName>')

# 删除对象标签
k.delete_object_tagging()
```

设置对象标签的更多详情，请参见 Delete Object Tagging。

数据加密

安装

- 使用npm安装sdk开发包，安装命令为npm install ks3 --save，如遇到网络问题，建议配置国内镜像地址。
- 使用方式：Node SDK目前仅支持异步方式，使用new KS3()创建client。
- Node SDK源码下载地址：点击下载。

初始化

使用密钥初始化

```
var KS3 = require('ks3');

//方式一 使用region创建
var client = new KS3('<AK>','<SK>','<BucketName>', '<Region>');

//方式二 不使用region时可使用下面的形式创建【推荐】
// var client = new KS3({
//     AK: '<ak>',
//     SK: '<sk>',
//     BucketName: '<bucketName>',
//     Endpoint: '<endpoint>'
// })
```

说明：如果您使用的Bucket所属Region是KS3提供，直接使用方式一即可。如果是其他区域的或者自定义域名方式访问可以使用方式二。

参数说明：

- AK：金山云提供的ACCESS KEY ID，必填
- SK：金山云提供的SECRET KEY ID，必填
- Bucket：存储空间的名称，非必填，也可在调用具体api时指定
- Region：存储空间所在的区域，非必填
- Endpoint：金山云提供的各个Region的域名（例 ks3-cn-beijing.ksyuncs.com），具体定义可参考 API接口文档-Region(区域)，也可以是用户自定义域名，如果是用户自定义域名，需要将domainMode设置为true，非必填
- DomainMode：是否支持自定义域名，默认值为false，非必填
- Internal：是否内网访问，默认false，具体定义可参考内网访问资源。非必填
- Timeout：超时时间，默认值为60000，单位为毫秒，非必填
- Secure：是否使用https，设置true则使用https，否则http，默认false，非必填

快速入门

上面初始化完成以后，就可以开始创建存储空间、下载文件、上传文件等相关操作了。

- 创建存储空间

以下代码用于创建存储空间。

```
client.bucket.put({
  Bucket: '<bucketName>' // bucketName必须符合要求
}, function (rerr, data, response, body) {
  console.log(response.statusCode) // 如果成功返回200，已存在返回409
})
```

- 列举存储空间

以下代码用于查看存储空间列表。

```
client.service.listBucket(function(rerr, data, response, body) {
  console.log(data)
})
```

- 获取文件列表

以下代码用于获取指定存储空间下的文件列表。

```
client.bucket.get({
    Bucket: '<bucketName>',
    Delimiter: '/',
    Prefix: 'img',
    MaxKeys: 1000 // 默认1000
}, function (rerr, data, response, body) {
  console.log(data)
})
```

- 上传文件

以下代码用于上传单个文件。

```
client.object.put({
  Bucket: '<bucketName>',
  Key: '<objectKey>',
  FilePath: '<filePath>'
}, function (rerr, data, response, body) {
  console.log(data)
})
```

- 下载文件

以下代码用于下载文件到本地。

```
client.object.get({
  Bucket: '<bucketName>',
  Key: '<objectKey>'
}, function (rerr, data, response, body) {
  console.log(data)
})
```

- 删除文件

以下代码用于删除服务端指定文件。

```
client.object.del({
  Bucket: '<bucketName>',
  Key: '<objectKey>'
}, function (rerr, data, response, body) {
   console.log(data)
})
```

存储空间

上传文件（Node.js）

- 简单上传

以下代码用于上传小于5G的文件到指定到的存储空间，上传的同时可设置ACL、storageClass。

```
client.object.put({
  Bucket: '<bucketName>',
  FilePath: '',// 待上传文件路径，必填
  Key: '<objectKey>', // 唯一的key，必填
  StorageClass: '', // 存储类型STANDARD/STANDARD_IA/ARCHIVE，非必填
  ACL: '',// 访问控制private/public-read，非必填
  headers: {}// 自定义header信息，非必填
}, function (rerr, data, response, body) {
  console.log(response.statusCode)// 成功返回200
})
```

- 分块上传

以下代码适用于上传超过5G的文件到指定存储空间。

分块上传分三步： 1. 初始化 2. 上传分块 3. 合并分块

```
// 1.初始化
client.object.multitpart_upload_init({
  Bucket: '<bucketName>',
  Key: '<objectKey>', // 必填
  ACL: '', // 访问控制，非必填
  StorageClass: '', // 存储类型，非必填
  headers: {} // 非必填
}, function (rerr, data, response, body) {
  console.log(data) // data中会包含后续上传需要的UploadId
})

// 2.上传分块
// notice： 上传分块前需要自行对文件进行分块处理
client.object.upload_part({
  Bucket: '<bucketName>',
  Key: '<objectKey>', // 必填
  PartNumber: '', // 必填, 上传的块id
  UploadId: '', // 必填, 上传的uploadid
  Body: '' // 必填, 分块数据
}, function (rerr, data, response, body) {
    console.log(response.statusCode) // 上传成功返回200
    console.log(response.headers.etag) // 上传成功返回etag，后面合并分块会用到
})

// 3.合并分块
client.object.upload_complete({
  Bucket: '<bucketName>',
  Key: '<objectKey>', // 必填
  UploadId: '', // 必填, 上传的uploadid
  Body: '' // 必填, 所有分块数据的集合，以xml格式传递
}, function (rerr, data, response, body) {
  console.log(data)
})
```

完整的示例：

```
function upload() {
    const bucket = '' // 上传的桶
    const key = '' // 上传的key
    const filePath = '' // 文件路径
    const chunkSize = 5*1024*1024 // 分块大小
    const contentType = '' // 文件类型
    let uploadId = '' // 上传id
    const fileSize = 100*1025*1024 // 文件大小
    const count = parseInt(fileSize / chunkSize) + ((fileSize % chunkSize == 0 ? 0: 1));     // 总块数

    const initParams = {
        Key: key,
        Bucket: bucket
    }
    client.object.multitpart_upload_init(initParams, function (err, data, res, body) {

        data = util.xml2json.parser(data) // util为sdk内部的工具类

        uploadId = data.InitiateMultipartUploadResult.UploadId
        console.log(`获取到上传id： ${uploadId}`)

        client.config({
            dataType: 'xml'
        });

        let chunkNum = 0;

        let etags = [] // 存放上传成功的分块etag

        console.log('开始分块上传...')

        up();

        function up() {
            let start = chunkNum * chunkSize
            console.log(`开始上传 第${chunkNum} 块文件...`)
            // 是否全部上传
            if (chunkNum < count) {
                util.getChunk(filePath, chunkSize, start, function (buffer) {
                    const params = {
                        Bucket: bucket,
                        Key: key,
                        UploadId: uploadId,
                        PartNumber: chunkNum + 1, // 传值从1开始
                        Body: buffer,
                        Type: contentType
                    }
```

```
                    console.log('分块上传参数：', params)
                    client.object.upload_part(params, function (err, data, res, body) {
                        if (err) {
                            throw err;
                        } else {
                            const etag = res.headers.etag;
                            console.log(`上传成功，获取etag信息：${etag}`)
                            etags.push(etag)
                            chunkNum++;
                            up();
                        }
                    })
                })
            } else {
                // 合并上传的分块
                const params = {
                    Bucket: bucket,
                    Key: key,
                    PartNumber: count,
                    Type: contentType,
                    UploadId: uploadId,
                    Body: util.generateCompleteXML(arr) // 将分块数据转成xml
                }
                console.log('合并参数：', params)
                client.object.upload_complete(params, function (err, data, res, body) {
                    if (err) throw err;
                    console.log('上传成功，合并文件：', err, data)
                })
            }
        }
    });
}
```

说明： 该完整示例没有对异常做处理，是种理想情况下的场景。建议是结合自身的业务调整逻辑。其他具体说明需要查看文档。

- 流式上传

```
client.object.put({
  Bucket: '<bucketName>',
  Key: '<objectKey>', // 唯一的key，必填
  Body: ''// stream，必填
}, function (rerr, data, response, body) {
  console.log(response.statusCode)// 成功返回200
})
```

- 上传回调

回调原理：

```
client.object.put({
  Bucket: '<bucketName>',
  Key: '<objectKey>', // 唯一的key，必填
  FilePath: '',// 待上传文件路径，必填
  headers: {
    'x-kss-callbackurl':"<应用服务器地址>", //回调地址
    'x-kss-callbackbody': 'etag=${etag}&objectSize=${objectSize}&key=${key}',
    'x-kss-callbackauth': '1',// 是否开启验证
    'kss-location': 'test'
  }
}, function (rerr, data, response, body) {
  console.log(response.statusCode)// 成功返回200
})
```

说明：

- headers内的回调参数未做校验，请确保传参正确
- callbackbody：支持自定义参数、常量、魔法变量。对于魔法变量可以简单理解为约定好的固定变量，使用时直接用${变量名}格式即可取到相应的值。常用的魔法变量参考文档说明。
- 如果callbackurl不可达，则会报错。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Error>
  <Code>CallbackFail</Code>
  <Message>call back server is fail</Message>
  <Resource>/bucketName/a.mp4?callback=''</Resource>
</Error>
```

管理文件

对象标签（Node.js）

- 设置对象标签

以下代码用于设置对象标签：

```
const tags = [{
  key: 's',
  value: 'asd'
}, {
  key: 'string1',
  value: 'string'
}]

client.object.putObjectTagging({
  Bucket: '<bucketName>',
  Key: '<objectKey>',// 必填
  Taggings: tags // 将想要配置的标签以json的形式传递，必填
}, function (rerr, data, response, body) {
  console.log(data)
})
```

说明：详细的标签规则和错误码，详见文档。

- 获取对象标签

以下代码用于获取对象标签：

```
client.object.getObjectTagging({
  Bucket: '<bucketName>',
  Key: '<objectKey>' // 必填
}, function (rerr, data, response, body) {
  console.log(data) // 如果没有权限则会返回403
})
```

- 删除对象标签

以下代码用于删除对象标签：

```
client.object.deleteObjectTagging({
  Bucket: '<bucketName>',
  Key: '<objectKey>' // 必填
}, function (rerr, data, response, body) {
  console.log(response.statusCode) // 删除成功返回204，没有权限则返回403
})
```

安装

相关资源

- JavaScript SDK源码下载地址： 点击下载。
- JavaScript SDK在线调用地址：在线调用。
- SDK 更新日志请参见： JavaScript SDK更新记录。
- SDK 常见问题请参见：常见问题 。

  旧版JS SDK源码、使用文档：点击查看 。

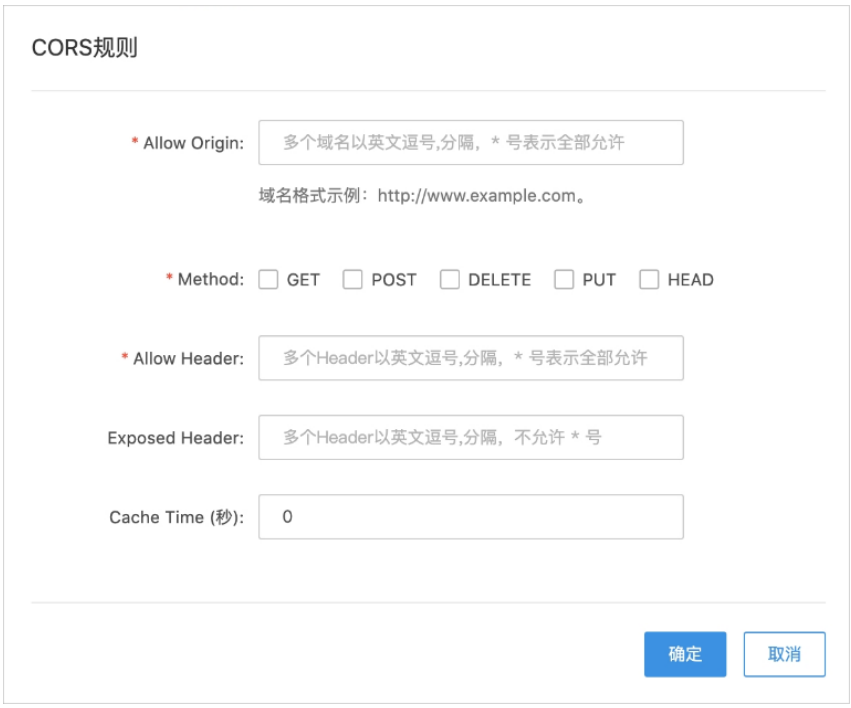
环境准备

1. JavaScript SDK 需浏览器支持基本的 HTML5 特性（支持 IE10 以上浏览器），以便支持 ajax 上传文件和计算文件 MD5 值。

2. 登录 KS3控制台 ，创建存储桶。获取存储桶名称和地域信息。

3. 登录访问控制（IAM） ，获取您的 AccessKeyID 和 SecretAccessKey，详细操作参见访问控制（IAM）文档。

4. 配置 CORS 规则，用户根据需求配置相应的参数值，如下图所示。操作详情请参见 CORS配置 文档。



安装

您可以通过以下方式安装SDK：

- 使用npm安装SDK
- 通过npm install ks3-js-sdk 安装 SDK ，支持 webpack 打包的场景，安装成功后即可通过import引入。 【推荐】

```
import KS3 from 'ks3-js-sdk';
```

- script引入

将SDK下载至本地，可通过script标签的方式引入。

```
<script src="./ks3-js-sdk.min.js"></script>
```

初始化

创建对象并传入配置项

```
import KS3 from 'ks3-js-sdk';

let config = {
    AK: '<ak>',
    SK: '<sk>',
    region: '<region>',
    bucket: '<bucket>'
}

let ks3 = new KS3(config);
```

config配置参数

名称	描述	是否必选
AK	您的AccessKeyID或者临时访问凭证STS的AccessKeyID	是
SK	您的SecretAccessKey或者临时访问凭证STS的SecretAccessKey	是
region	存储空间所在的区域，示例：BEIJING	否
domain	是否用户自定义域名，默认为false	否
bucket	存储空间的名称，也可在调用具体api时指定	否
protocol	自定义的请求协议，可选项 https:、http:，默认判断当前页面是 http: 时使用 http:，否则使用 https:	否

forceHostStyle	是否使用三级域名，默认值为true	否
chunkSize	分块上传的最小单位，默认值为5 *1024* 1024，单位为Byte	否
retryTimes	分块上传重试次数，默认值为2	否
parallelLimit	分块上传并发个数，默认值为10	否
progressInterval	回调时间间隔，默认值为1000，单位为毫秒	否
timeout	超时时长，默认值为0，表示不设置超时时间，单位为毫秒	否
securityToken	从STS服务获取的临时身份凭证（SecurityToken）	否

说明

- 如果您使用的bucket所属region是KS3提供，需要将domain设置为false。如果是用户自定义域名，需要将domain设置为true。
- 由于固定密钥放在前端会有安全风险，正式部署时我们推荐使用临时密钥的方式。实现过程为：前端首先请求服务端，服务端使用固定密钥调用 STS 服务申请临时密钥，然后返回临时密钥到前端使用。详情请参见 临时密钥生成文档。同时ks3 js sdk为您提供获取临时密钥示例，示例参见js sdk 内 example/server。

快速入门

初始化完成后，可以进行上传文件、下载文件、删除文件等相关操作。

注意

- 以下代码示例默认在初始化时已经进行了AK、SK、region、bucket的配置。
- 在实际请求中，会优先使用参数中的bucket和region，如果没有传递该参数，则会使用ks3初始化时config中的bucket和region。

获取文件列表

以下代码用于获取指定存储空间下的文件列表。

```
 ks3.listObjects({
    'delimiter':'/', // 用于对象名称进行分组的字符
    'max-keys': '500',// 每页最大数量，默认1000
    'prefix':'/'// 指定返回对象名的前缀
}.then(res => {
    console.log('listObjects:', res);
})
```

上传文件

以下代码用于上传单个文件。

```
ks3.putObject({
    key: 'demo.txt',    // 填写Object完整路径，必填
    file: 'file',       // 上传的文件，必填
    acl: 'private'      // 文件预设ACL(private|public-read)，默认值private，非必填
}.then(res => {
    console.log('putObject:', res);
})
```

下载文件

以下代码用于下载单个文件。

```
ks3.getObject({
    key: 'demo.txt'       // 填写Object完整路径，必填
}.then(res => {
    console.log('getObject:', res);
})
```

删除文件

以下代码用于删除指定文件。

```
ks3.delObject({
    key: 'demo.txt'       // 填写Object完整路径，必填
}.then(res => {
```

```
    console.log('delObject:', res);
})
```

上传文件

下载文件

获取对象

getObject - 用于获取指定对象的内容。

使用示例

```
ks3.getObject({
    key: 'demo.txt',            // 填写Object完整路径，必填
}.then(res => {
    console.log('getObject:', res);
})
```

生成对象预签名链接

getObjectUrl - 用于获取指定对象的Url。

使用示例

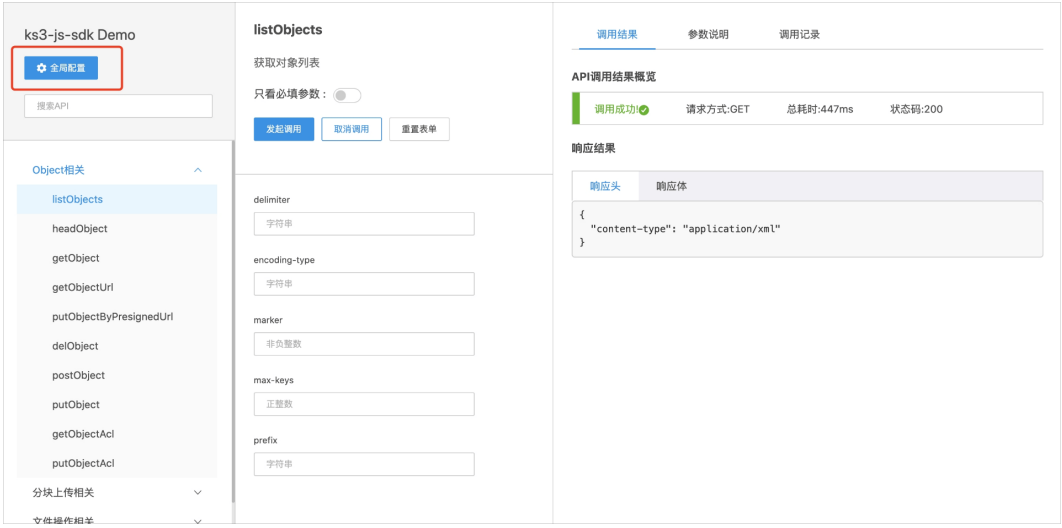
```
function getObjectUrl(){
    ks3.getObjectUrl({
        'key': 'IMG_7DBACC50C65A-1.jpeg',
        'expires': 5*60,        //签名几秒后失效，默认900秒，非必须
        'method': 'GET'        //请求方法，非必须,PUT/GET/POST/DELETE
    }).then(res => {
        console.log(res);
    });
}
```

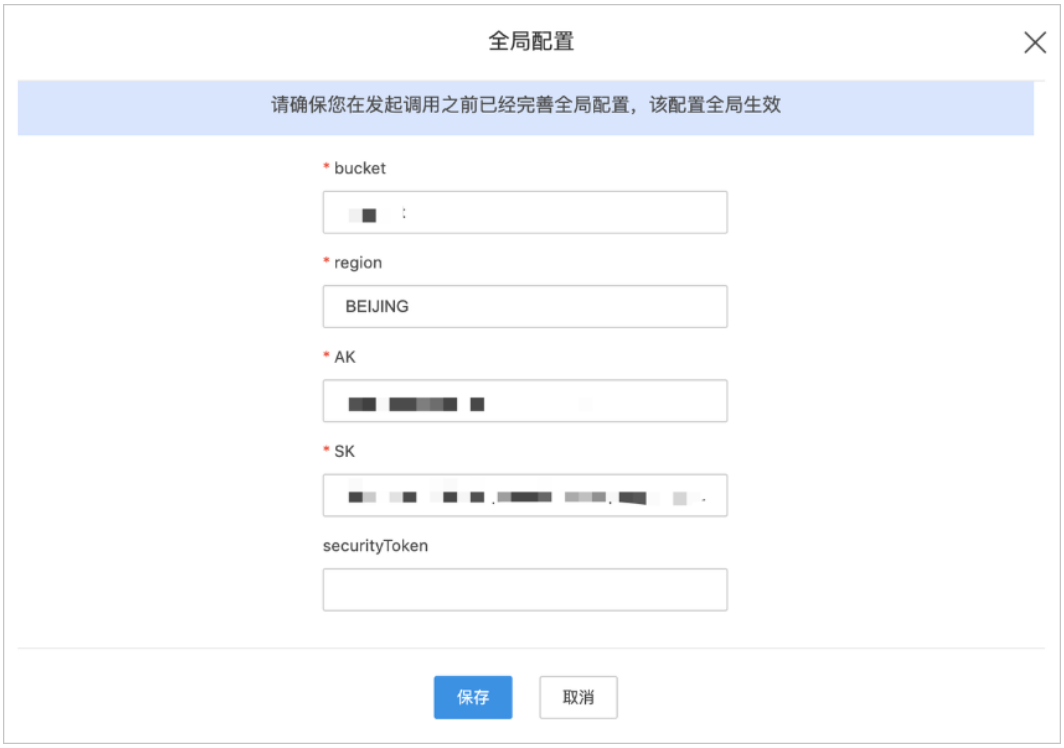
管理文件

Demo示例程序

Js sdk为方便用户调试，提供了demo文件，以下是操作流程:

1. 进入demo文件内，执行npm install;
2. 执行npm run serve;
3. 点击全局配置，配置必填信息;
4. 选择待调试的接口，填写对应参数，点击发起调用;
5. 右侧展示对应的调用结果和调用记录;





常见问题

---

1. 请给我一份适用于KS3 Javascript SDK policy文件示例

```
let policy = {
    expiration: "2022-10-02T10:57:30.000Z",
    conditions: [
```

```
        ["eq", "$bucket", bucket],
        ['eq', '$key', key],
        ['eq', '$x-kss-server-side-encryption', 'AES256']
    ]
}
```

2. 已经在ks3.js配置过的bucket或region，调用某个方法想特殊配置的话，每次都要修改ks3.js配置文件吗？

答：不需要，调用方法时若传参bucket，region则不读取配置文件的值；

```
function putObjectAcl(){
    ks3.putObjectAcl({
        key: '57M.pdf',
        acl: 'private',
        bucket: 'xxx',
        region: 'SHANGHAI'
    }).then(res => {
        logResult('putObjectAcl:', res);
    })
}
```

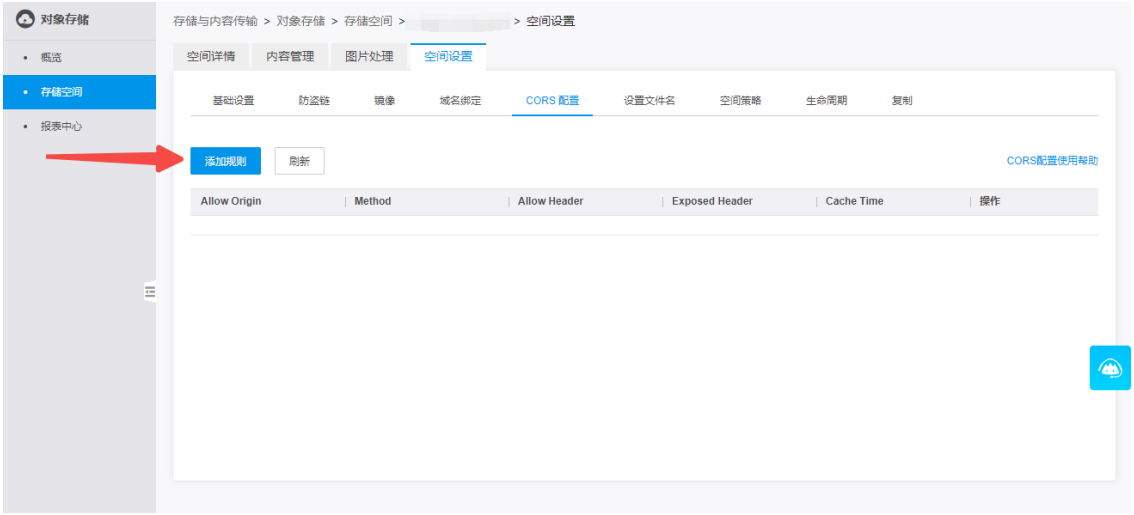
```
async function putObjectAcl(params){
    if (params.key === null || params.key === undefined) {
        throw Error('require the params of key!');
    }
    if(!params.bucket && !this.config.bucket){
        throw Error('require the params of bucket or this.config of bucket!');
    }
    if(!params.region && !this.config.region){
        throw Error('require the params of region or this.config of region!');
    }
    if(params.acl !== 'private' && params.acl !== 'public-read'){
        throw Error('The value of acl can only be private or public-read!');
    }
    var headers = params.headers || {};
    var resource = `?acl`;
    var bucket = params.bucket || this.config.bucket;
    var region = params.region |this.config.region;
    var url = getUrl.bind(this)({
        bucket,
```

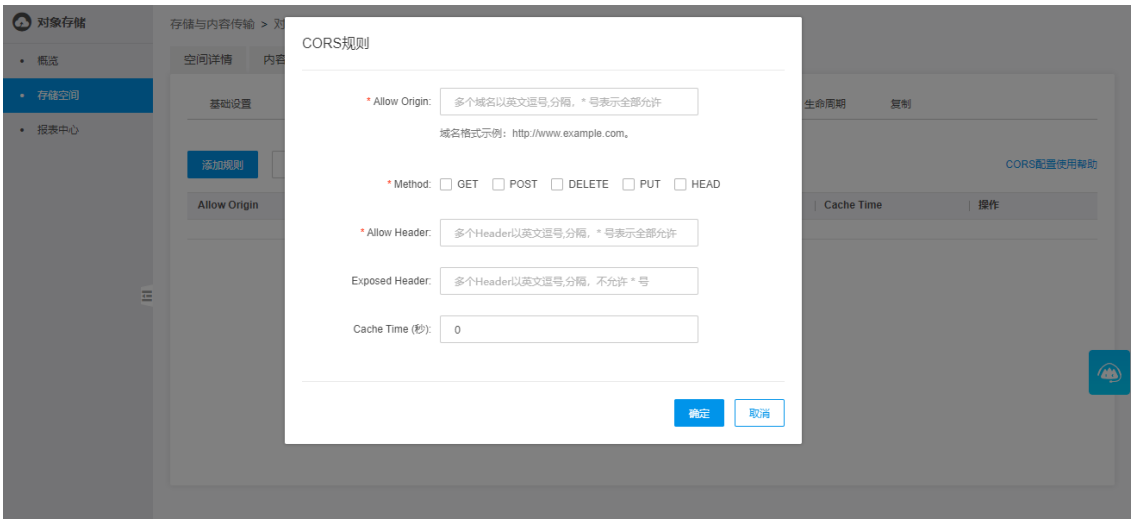
3. 使用SDK时报跨域错误怎么办？

```
Access to XMLHttpRequest at 'http://test-bucket.ks3-cn-beijing.ksyuncs.com/b' from origin 'http://127.0.0.1:5501' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

答：需要在控制台为您要操作的bucket配置跨域，具体操作如下：

- 在控制台进入您要配置跨域的桶，选择空间设置，选择CORS配置，点击添加规则。





4. 上传文件时对文件的大小有限制吗？

答：简单上传（putObject方法）仅支持上传5G以内大小的文件，若您要上传更大的文件，请使用分块上传（uploadMultiPart方法）。

5. 我如何获取上传文件的进度？

对于putObject和uploadMultiPart方法，可以传递progress参数获取实时上传进度，该参数的类型是Function，使用示例如下：

```
ks3.putObject({
    key: 'demo.txt',    // 填写Object完整路径，必填
    file: '<file>',     // 上传的文件，必填
    progress: function(e) {    // 上传进度监听函数，非必填
        console.log(e);
    }
}.then(res => {
    console.log('putObject res:', res);
})
```

6. 上传速度达不到满载带宽怎么办？

答：可以使用分块上传（uploadMultiPart方法），并适当增大分块的大小。SDK默认分块大小为5MB，您可以尝试将其调整为20MB。

7. SDK默认的超时时间是多少？

答：SDK默认不设置超时时间，若您有需要，可以在配置项中设置。

8. 如何将文件上传到指定目录中？

答：若您想在aaa目录下上传demo.txt文件，只需要将key值设为 aaa/demo.txt，若该目录不存在，则会自动创建目录。代

码示例如下：

```
ks3.putObject({
    key: 'aaa/demo.txt',    // 填写Object完整路径，必填
    file: '<file>',         // 上传的文件，必填
}.then(res => {
    console.log('putObject res:', res);
})
```

PHP

KS3 SDK For PHP使用指南

目录

---

1. 概述

此SDK适用于PHP 5及以上版本。基于KS3 API 构建。使用此 SDK 构建您的网络应用程序，能让您以非常便捷地方式将数据安全地存储到金山云存储上。

2. 环境准备

- 配置PHP 5 以上开发环境
- 添加curl拓展
- 下载KS3 SDK For PHP
- 在项目中引用该php文件，Ks3Client.class.php

3. 初始化

3.1 下载SDK

- 点击下载

3.2 获取密钥

- 开通KS3服务，https://www.ksyun.com/user/register 注册账号
- 进入控制台，https://ks3.console.ksyun.com/console.html#/setting 获取AccessKeyID 、AccessKeySecret

3.3 配置

在引用Ks3Client.class.php文件前定义：

- //是否使用VHOST
  define("KS3_API_VHOST",TRUE);

- //是否开启日志(写入日志文件)
  define("KS3_API_LOG",TRUE);
- //是否显示日志(直接输出日志)
  define("KS3_API_DISPLAY_LOG", TRUE);
- //定义日志目录(默认是该项目log下)
  define("KS3_API_LOG_PATH","");
- //是否使用HTTPS
  define("KS3_API_USE_HTTPS",FALSE);
- //是否开启curl debug模式
  define("KS3_API_DEBUG_MODE",FALSE);

3.4 初始化客户端

当以上全部完成之后用户便可初始化客户端进行操作了：

- $client = new Ks3Client("<您的AccessKeyID>","<您的AccessKeySecret>","endpoint")

endpoint与Region对应关系

3.5 常见术语介绍

Object（对象，文件）

在KS3中，用户操作的基本数据单元是Object。单个Object允许存储0~48.8TB的数据。 Object 包含key和data。其中，key是Object的名字；data是Object 的数据。key为UTF-8编码，且编码后的长度不得超过1024个字符。

Key（文件名）

即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字符。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

其他常见术语请参考概念与术语

4. 异常说明

4.1 Ks3ServiceException

当抛出Ks3ServiceException时表示KS3服务端返回异常信息。Ks3ServiceException继承自RuntimeException 用户可根据该异常中的信息获取到出错的原因。

4.2 Ks3ClientException

当抛出Ks3ClientException时表示客户端发送了异常。Ks3ClientException继承自RuntimeException。

5. 使用示例

快速导航：

- 删除文件：5.3.1 5.3.2
- 下载文件：5.3.3
- 判断文件是否存在：5.3.5
- 上传文件：5.3.7
- 分块上传：5.3.12-5.3.17

5.1 Service接口

5.1.1 GET Service

获取用户的所有bucket列表
使用示例：

```
$client->listBuckets();
```

返回结果格式：

```
Array
(
    [0] => Array
    (
        [Name] => aaphp
        [CreationDate] => 2015-03-21T06:25:45.000Z
    )

    [1] => Array
    (
        [Name] => adest
        [CreationDate] => 2015-02-10T03:55:40.000Z
    )

    [2] => Array
    (
        [Name] => afiles
        [CreationDate] => 2015-02-10T07:39:19.000Z
    )
)
```

5.2 Bucket接口

- Delete Bucket 删除Bucket
- DELETE Bucket cors 删除Bucket的跨越配置
- Get Bucket 罗列Bucket下的object
- GET Bucket acl 获取bucket的权限
- GET Bucket cors 获取bucket的跨域配置
- GET Bucket location 获取bucket的地址
- GET Bucket logging 获取bucket的日志配置
- HEAD Bucket 判断一个bucket是否存在
- List Mutipart Uploads列出当前bucket下未完成的分块上传
- PUT Bucket 创建一个bucket
- PUT Bucket acl 设置bucket的访问权限
- PUT Bucket cors 设置bucket的跨域规则

- PUT Bucket logging设置bucket的日志配置

5.2.1 DELETE Bucket

删除Bucket
注意：
1、只能删除空的Bucket
参数格式：

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例：

```
$client->deleteBucket($args);
```

5.2.2 DELETE Bucket cors

删除Bucket的跨域配置
参数格式：

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例：

```
$client->deleteBucketCORS($args);
```

5.2.3 GET Bucket

罗列Bucket下的object
参数格式：
Options中为可选参数，用户需参考KS3 API文档根据实际情况调节参数。

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Options"=>array(
        "prefix"=>"<prefix>",
        "max-keys"=>"<max-keys>",
        "marker"=>"<marker>",
```

```
        "delimiter"=>"<delimiter>"
    )
);
```

使用示例：

```
$client->listObjects($args);
```

返回结果格式：

```
Array
(
    [Name] => ksc-scm //bucket名称
    [Prefix] =>
    [Marker] =>
    [Delimiter] => /
    [MaxKeys] => 4
    [IsTruncated] => true  //true表示返回的结果是全部结果的一部分
    [NextMarker] => dir/  //如果IsTruncated为true，则可以使用NextMarker作为下一次请求的marker.当请求时不提供delimiter的话不会返回Ne
xtMarker,可以使用Contents的最后一项作为下一次的Marker
    [Contents] => Array
    (
        [0] => Array
            (
                [Key] => 123.pdf
                [LastModified] => 2015-02-06T07:39:32.687Z
                [ETag] => 1285ba0d89e9b0883a1a5975051af159
                [Size] => 515602
                [Owner] => Array
                    (
                        [ID] => 73398334
                        [DisplayName] => 73398334
                    )

                [StorageClass] => STANDARD
            )

        [1] => Array
            (
                [Key] => 20150210154319.jpg
                [LastModified] => 2015-02-10T07:44:20.818Z
                [ETag] => c61d3bbb47947029b968d02be1cae7d0
                [Size] => 141179
                [Owner] => Array
                    (
                        [ID] => 73398334
                        [DisplayName] => 73398334
                    )

                [StorageClass] => STANDARD
            )

        [2] => Array
            (
                [Key] => chrome.exe
                [LastModified] => 2015-01-07T05:30:26.845Z
                [ETag] => ac08a03d7e579e2903925736e7ab48f2
                [Size] => 852808
                [Owner] => Array
                    (
                        [ID] => 73398334
                        [DisplayName] => 73398334
                    )

                [StorageClass] => STANDARD
            )

    )

    [CommonPrefixes] => Array
    (
        [0] => dir/
    )
)
```

5.2.4 GET Bucket acl

获取bucket的权限
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例:

```
$client->getBucketAcl($args);
```

返回结果:

```
private、public-read或者public-read-write
```

5.2.5 GET Bucket cors

获取bucket的跨域配置
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例

```
$client->getBucketCORS($args);
```

返回结果格式:

```
Array
(
    [0] => Array
        (
            [AllowedOrigin] => Array
            (
                [0] => http://www.kingsoft.com
            )
            [AllowedMethod] => Array
            (
                [0] => GET
                [1] => PUT
            )
            [AllowedHeader] => Array
            (
                [0] => *
            )
            [MaxAgeSeconds] => 10
            [ExposeHeader] => Array
            (
                [0] => *
            )
        )
    [1] => Array
        (
            [AllowedOrigin] => Array
            (
                [0] => *
            )
            [AllowedMethod] => Array
            (
                [0] => GET
                [1] => PUT
            )
            [AllowedHeader] => Array
            (
                [0] => *
            )
            [MaxAgeSeconds] => 10
            [ExposeHeader] => Array
            (
                [0] => *
            )
        )
)
```

5.2.6 GET Bucket location

获取bucket的地址
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例:

```
$client->getBucketLocation($args);
```

返回结果格式:

```
HANGZHOU
```

5.2.7 GET Bucket logging

获取bucket的日志配置
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例:

```
 $client->getBucketLogging($args);
```

返回结果格式:

```
Array
(
    [Enable] => 1
    [TargetBucket] => ksc-scm
    [TargetPrefix] =>
)
```

5.2.8 HEAD Bucket

判断一个bucket是否存在
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>")
```

使用示例:

```
$client->bucketExists($args);
```

返回结果格式:

```
TRUE或者FALSE
```

5.2.9 List Mutipart Uploads

列出当前bucket下未完成的分块上传
参数格式:

```
$args=array(
    "Bucket"=><目标bucket名称>,
    "Options"=>array(
        "max-uploads"=>1, //调节参数,支持key-marker、prefix、upload-id- marker、delimiter,详细作用请参考KS3 API文档
    )
)
```

使用示例:

```
$client->listMutipartUploads($args);
```

返回结果格式:

```
Array
(
    [Bucket] => phpsdktestlijunwei
    [KeyMarker] =>
    [UploadIdMarker] =>
    [NextKeyMarker] =>
    [NextUploadIdMarker] =>
    [MaxUploads] => 1
    [IsTruncated] => true //true表示返回的结果只是全部结果的部分
    [Uploads] => Array
    (
        [0] => Array
        (
            [Key] => dir/test/中文目录/@特殊字符!.txt
            [Initiated] => 2015-03-23T11:22:45.451+08:00
            [UploadId] => b05e21c69ff14386a66bbe9843976b17
            [Initiator] => Array
            (
                [ID] => 73398529
                [DisplayName] => 73398529
            )
            [Owner] => Array
```

```
            (
                [ID] => 73398529
                [DisplayName] => 73398529
            )
            [StorageClass] => STANDARD
        )
    )
)
```

5.2.10 PUT Bucket

创建一个bucket
注意:
1、bucket名称是全局唯一，如果返回BucketAlreadyExists请尝试换一个名称
2、bucket名称规则请参考KS3 API文档
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "ACL"=>"private", //配置bucket的访问权限，合法值:private、public-read、public-read-write
    "Location"=>"BEIJING"//配置bucket存储地址，默认是北京
```

。BEIJING,SHANGHAI,HONGKONG)

使用示例:

```
$client->createBucket($args);
```

5.2.11 PUT Bucket acl

设置bucket的访问权限
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "ACL"=>"private" //配置bucket的访问权限，合法值:private、public-read、public-read-write
    )
```

使用示例:

```
$client->setBucketAcl ($args);
```

5.2.12 PUT Bucket cors

设置bucket的跨域规则
注意:
1、如果返回InvalidArguments，请查看KS3 API文档，查看各参数格式要求。
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "CORS"=>array(
        //设置第一条规则
        array(
        //指定允许的跨域请求方法为GET和PUT (支持GET/PUT/DELETE/POST/HEAD)
            "AllowedMethod"=>array("GET","PUT"),
        //指定允许跨域请求的来源为http://www.kingsoft.com
            "AllowedOrigin"=>array("http://www.kingsoft.com"),
        //允许所有响应头的跨域请求
            "AllowedHeader"=>array("*"),
        //允许用户从应用程序中访问所有的响应头
            "ExposeHeader"=>array("*"),
        //设置浏览器缓存该响应的时间为10s
            "MaxAgeSeconds"=>10
        ),
        //设置另一条规则
        array(
        //指定允许的跨域请求方法为GET和PUT (支持GET/PUT/DELETE/POST/HEAD)
            "AllowedMethod"=>array("GET","PUT"),
        //允许所有源的跨域请求
            "AllowedOrigin"=>array("*"),
        //允许所有响应头的跨域请求
            "AllowedHeader"=>array("*"),
        //允许用户从应用程序中访问所有的响应头
            "ExposeHeader"=>array("*"),
        //设置浏览器缓存该响应的时间为10s
            "MaxAgeSeconds"=>10
```

```
        )
    )
);
```

使用示例

```
$client->setBucketCORS($args);
```

5.2.13 PUT Bucket logging

设置bucket的日志配置
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "BucketLogging"=>array(
        "Enable"=>TRUE,//是否开启
        "TargetBucket"=>"ksc-scm",
        "TargetPrefix"=>"X-KSS"
        )
    );
```

使用示例

```
$client->setBucketLogging ($args);
```

5.3 Object接口

- DELETE Object 删除一个object
- DELETE Multiple Objects删除多个object
- GET Object 下载object
- GET Object acl 获取object的权限
- HEAD Object 判断object是否存在或获取object的元数据
- POST Object 表单上传文件
- PUT Object 上传文件
- PUT Object acl 设置object的访问权限
- PUT Object - Copy object拷贝相关
- Initiate Multipart Upload 分块上传
- Upload Part 上传块
- List Parts 列出一个分块上传已经上传的块
- Complete Multipart Upload 完成分块上传
- Abort Multipart Upload 取消分块上传
- Mutipart Upload(分块上传) Demo 分块上传示例
- PUT Object - rename 文件重命名
- 使用外链操作

5.3.1 DELETE Object

删除一个object
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"123.pdf"
    );
```

使用示例:

```
$client->deleteObject($args);
```

5.3.2 DELETE Multiple Objects

删除多个object

此接口为异步执行接口，调用成功后接口会快速返回结果，但是删除任务会在后台异步执行，即不保证删除的实时性，如需实时删除请调用DELETE Object接口一个一个删除

参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "DeleteKeys"=>array("<key1>","<key2>","<key3>")
    );
```

使用示例:

```
$client->deleteObjects($args);
```

5.3.3 GET Object

5.3.3.1 下载object

参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Range"=>array( //当需要分块下载时可以提供该参数，start为起始位置，end为结束位置，单位为Byte,0表示第一个字节，可根据需求设置起始
、结束位置
        "start"=>0,//请求内容的前四个字节
        "end"=>3,//
        ),
    "WriteTo"=>"D://test.zip" //文件保存路径,可以不提供。可以是resource
    );
```

使用示例:

```
$client->getObject($args);
```

返回结果格式(当不提供WriteTo时会有该返回结果):

```
Array
(
    [Content] => "1234"//文件内容
    [Meta] => Array
    (
        [ObjectMeta] => Array   //元数据
        (
            [Content-Type] => binay/ocet-stream
            [Content-Length] => 4
            [ETag] => "81dc9bdb52d04dc20036dbd8313ed055"
            [Last-Modified] => Sat, 21 Mar 2015 06:31:28 GMT
        )
        [UserMeta] => Array    //用户自定义元数据
        (
            [x-kss-meta-test] => test
        )
    )
)
```

5.3.3.2 下载经过客户提供主密钥的服务端加密数据

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<PUT Object时使用的主密钥>",//
    "KeyBase64"=>"<PUT Object时使用的主密钥的Base64值>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<PUT Object时使用的主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.3.3 生成object外链

参数格式:

```
$args=array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Options"=>array(
        "Expires"=>60*60*24*10,//过期时间，单位秒，即x秒后过期
        "response-content-type"=>"application/xml"//覆盖返回的http header,支持的值"response-expires","response-content-encoding","
response-content-disposition","response-content-language","response-content-type","response-cache-control"
        )
    );
```

使用示例:

```
$client->generatePresignedUrl($args);
```

返回结果格式:

http://aaphp.ks3-cn-beijing.ksyuncs.com/multi.zip?Expires=1427900010&response-content-type=application%2Fxml&KSSAccessKeyId=2HITWMQXL2VBB3XMAEHQ&Signature=E3YAKqMpO%2BVoBaslu%2B3eE3Ki97w%3D

5.3.4 GET Object acl

获取object的权限
参数格式:

```
$args = array("Bucket"=>"<您的bucket名称>","Key"=>"<key>")
```

使用示例:

```
$client->getObjectAcl($args);
```

返回结果:

```
private、public-read或者public-read-write
```

5.3.5 HEAD Object

可用来判断object是否存在或者获取object的元数据
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>"
    );
```

5.3.5.1 判断object是否存在

使用示例:

```
$client->objectExists ($args);
```

返回结果格式:

```
TRUE或者FALSE
```

5.3.5.2 获取object元数据

使用示例:

```
$client->getObjectMeta($args);
```

返回结果格式:

```
Array
(
    [ObjectMeta] => Array   //元数据
    (
        [Content-Type] => binay/ocet-stream
        [Content-Length] => 4
        [ETag] => "81dc9bdb52d04dc20036dbd8313ed055"
        [Last-Modified] => Sat, 21 Mar 2015 06:31:28 GMT
    )
    [UserMeta] => Array    //用户自定义元数据
    (
        [x-kss-meta-test] => test
    )
)
```

5.3.5.3 请求经过客户提供主密钥的服务端加密数据

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<PUT Object时使用的主密钥>",//
    "KeyBase64"=>"<PUT Object时使用的主密钥的Base64值>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<PUT Object时使用的主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.6 POST Object

表单上传文件，用于获取KSSAccessKeyId、Policy和Signature
使用示例:

```
$result = postObject(
            $bucket ,//要上传的bucket
            $postFormData=array(),//表单中能确定值得表单项, 如 array("key"=>"123.txt");
            $unknowValueFormFiled=array(),//表单中不能确定值得表单项,如 array("random1","random2");
            $filename=NULL,//如果没有使用${filename}占位符的话可以不提供要上传的文件名
            $expire=18000//生成的签名过期时间,单位秒
        );
```

返回结果格式:

```
Array
(
    [Policy] => eyJleHBpcmF0aW9uIjoiMjAxNS0wMy0yNFQwMTozMjo0Ny4yODhaIiwiY29uZGl0aW9ucyI6W3sia2V5IjoiZm9ybXVwbG9hZHRlc3RcL0BcdTRlMmRcdTY1ODcudHh0In0seyJzdWNjZXNzX2FjdGlvbl9yZWRpcmVjdCI6ImhOdHA6XC9cL2tzcy5rc311bi5jb21cL3BocHNka3Rlc3RsaWp1bndlaSJ9LHsiQ29udGVudC1UeXBlIjoidGV4dFwvaHRtbCJ9LHsieYnVja2V0IjoicGhwc2RrdGVzdGxpanVud2VpIn0sWyJzdGFydHMtd2l0aCIsIiRyYW5kb20iLCIiXV19
    [Signature] => yQcB+sUpyjpwZu2JN9KYf2kXPCI=
    [KSSAccessKeyId] => 1GLO2rRYQxK8s7FQh8dV
)
```

5.3.7 PUT Object

上传文件

5.3.7.1 通过内容上传

参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Content"=>"1234",//要上传的内容
    "ACL"=>"public-read",//可以设置访问权限,合法值,private、public-read
    "ObjectMeta"=>array(//设置object的元数据,可以设置"Cache-Control","Content-Disposition","Content-Encoding","Content-Length","Content-MD5","Content-Type","Expires"。当设置了Content-Length时，请勿大于实际长度，如果小于实际长度，将只上传部分内容。
        "Content-Type"=>"binay/ocet-stream",
        "Content-Length"=>4
        ),
    "UserMeta"=>array(//可以设置object的用户元数据，需要以x-kss-meta-开头
        "x-kss-meta-test"=>"test"
        )
    )
```

使用示例:

```
$client->putObjectByContent($args);
```

返回结果格式:

```
Array
(
    [ETag] => "????"
)
```

5.3.7.2 通过文件上传

参数格式:

```
$content = fopen("<文件路径>", "r");
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Content"=>array(//要上传的内容
        "content"=>$content,//可以是文件路径或者resource,如果文件大于2G，请提供文件路径
        "seek_position"=>0//跳过文件开头?个字节
        ),
    "ACL"=>"public-read",//可以设置访问权限,合法值,private、public-read
    "ObjectMeta"=>array(//设置object的元数据,可以设置"Cache-Control","Content-Disposition","Content-Encoding","Content-Length","Content-MD5","Content-Type","Expires"。当设置了Content-Length时，最后上传的为从seek_position开始向后Content-Length个字节的内容。当设置了Content-MD5时，系统会在服务端进行md5校验。
        "Content-Type"=>"binay/ocet-stream"
        //"Content-Length"=>4
        ),
    "UserMeta"=>array(//可以设置object的用户元数据，需要以x-kss-meta-开头
        "x-kss-meta-test"=>"test"
        )
    );
```

使用示例:

```
$client->putObjectByFile ($args);
```

返回结果格式:

```
Array
(
    [ETag] => "????"
)
```

5.3.7.3 上传文件时使用服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSE"=>array(
    "Algm"=>"<服务端加密算法>"//暂时支持AES256
)
```

5.3.7.4 上传文件时使用客户提供主密钥的服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<主密钥>",//KS3服务端将使用该主密钥对数据进行加密
    "KeyBase64"=>"<主密钥的Base64>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.8 PUT Object acl

设置object的访问权限
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "ACL"=>"private" //合法值 private、public-read
    );
```

使用示例:

```
$client->setObjectAcl ($args);
```

5.3.9 PUT Object - Copy

5.3.9.1 基本方法

拷贝object
参数格式:

```
$args = array(
    "Bucket"=>"<目标bucket>",
    "Key"=>"<目标key>",
    "CopySource"=>array(
        "Bucket"=>"<源bucket>",
        "Key"=>"<源key>"
        )
    );
```

使用示例:

```
$client->copyObject ($args);
```

5.3.9.2 被拷贝的Object是经过客户提供主密钥服务端加密的

在原有参数的基础上添加

```
"SSECSource"=>array(
    "Algm"=>"AES256",
    "Key"=>"<主密钥>",//KS3服务端将使用该主密钥对数据进行解密
    "KeyBase64"=>"<主密钥的Base64>",//Key和KeyBase64提供一个即可
```

```
    "KeyMD5"=>"<主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.9.3 Copy后的Object使用服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSE"=>array(
    "Algm"=>"<服务端加密算法>"//暂时支持AES256
)
```

5.3.9.4 Copy后的Object使用客户提供主密钥的服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<主密钥>",//KS3服务端将使用该主密钥对数据进行加密
    "KeyBase64"=>"<主密钥的Base64>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.10 Initiate Multipart Upload

5.3.10.1 基本方式

初始化分块上传
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "UserMeta"=>array(//可以设置用户元数据,需要以x-kss-meta-开头
        "x-kss-meta-test"=>"example"
        ),
    "ObjectMeta"=>array(//可以设置用户元数据,合法值,"Cache-Control","Content-Disposition","Content-Encoding","Content-Type","Expir
es"
        "Content-Type"=>"text/plain"
        )
    );
```

使用示例:

```
$client->initMultipartUpload ($args);
```

返回结果格式:

```
Array
(
    [Bucket] => aaphp
    [Key] => multi.zip
    [UploadId] => bbdb766a65ef43ebad2b2531739092d0
)
```

5.3.10.2 使用服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSE"=>array(
    "Algm"=>"<服务端加密算法>"//暂时支持AES256
)
```

5.3.10.3 使用客户提供主密钥的服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<主密钥>",//KS3服务端将使用该主密钥对数据进行加密
    "KeyBase64"=>"<主密钥的Base64>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
```

```
    )
```

5.3.11 Upload Part

5.3.11.1 基本方式

上传块
参数格式:
主要通过seek_position和Content-Length参数控制上传的内容范围

```
$args=array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "LastPart"=>FALSE,//指定当前块是否为最后一块
    "Options"=>array(
        "partNumber"=><partNumber>,//当前上传块的序号，需要为连续的正整数，
        "uploadId"=><uploadId>//由Initiate Multipart Upload获得
        ),
    "ObjectMeta"=>array(
        "Content-Length"=>min($partsize, $total - $partsize * $i)//每次上传$partsize大小（最后一块传剩余大小）
        "Content-MD5"=>"<Content-MD5>"//可以提供该块的MD5值，将在服务端进行MD5校验
        ),
    "Content"=>array(
        "content"=><file>,//要上传的文件路径,可以为resource,如果文件大于2G，请提供文件路径
        "seek_position"=>//跳过文件开头?个字节
        )
    );
```

使用示例:

```
$client->uploadPart ($args);
```

返回结果格式:

```
Array
(
    [ETag] => "9430d3a88773837eed6ce0f136770ea3"
)
```

5.3.11.2 使用客户提供主密钥的服务端加密

参数格式:
在原有参数的基础上加上如下

```
"SSEC"=>array(
    "Algm"=>"AES256",
    "Key"=>"<主密钥>",//KS3服务端将使用该主密钥对数据进行加密
    "KeyBase64"=>"<主密钥的Base64>",//Key和KeyBase64提供一个即可
    "KeyMD5"=>"<主密钥经Base64编码的MD5值>",//可以不指定，SDK将根据Key计算
    )
```

5.3.12 List Parts

列出一个分块上传已经上传的块
参数格式:

```
$args = array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Options"=>array("uploadId"=><uploadId>)//由Initiate Multipart Upload获得
)
```

使用示例:

```
$client->listParts($args);
```

返回结果格式:

```
Array
(
    [Bucket] => aaphp
    [Key] => multi.zip
    [UploadId] => bbdb766a65ef43ebad2b2531739092d0
    [StorageClass] =>
    [PartNumberMarker] => 1
    [NextPartNumberMarker] =>
    [MaxParts] => 1000
```

```
    [IsTruncated] => false
    [Owner] => Array
    (
        [ID] => NzMzOTgzMzQ=
        [DisplayName] => NzMzOTgzMzQ=
    )
    [Initiator] => Array
    (
        [ID] => NzMzOTgzMzQ=
        [DisplayName] => NzMzOTgzMzQ=
    )

    [Parts] => Array
    (
        [0] => Array
            (
                [PartNumber] => 1
                [ETag] => 9430d3a88773837eed6ce0f136770ea3
                [LastModified] => 2015-03-22T12:48:32.800Z
                [Size] => 5242880
            )

        [1] => Array
            (
                [PartNumber] => 2
                [ETag] => cd18e9d26dba2121baf291693158b84b
                [LastModified] => 2015-03-22T12:48:54.716Z
                [Size] => 5242880
            )

        [2] => Array
            (
                [PartNumber] => 3
                [ETag] => bfd9044ef806a7408dd6ae803654f1a0
                [LastModified] => 2015-03-22T12:49:17.163Z
                [Size] => 5242880
            )

        [3] => Array
            (
                [PartNumber] => 4
                [ETag] => 36a2f4d8582cbd37a20c1541d0aff1cd
                [LastModified] => 2015-03-22T12:49:17.390Z
                [Size] => 37896
            )
    )
)
```

5.3.13 Complete Multipart Upload

5.3.13.1 基本方式

完成分块上传
参数格式:

```
$args=array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Options"=>array("uploadId"=><uploadId>)//由Initiate Multipart Upload获得
    "Parts"=>array(//需要提供Upload Part时的PartNumber和ETag。可以直接使用List Parts的返回结果中的Parts
        array(
            "PartNumber"=>1,
            "ETag"=>"7778aef83f66abc1fa1e8477f296d394"
            ),
        array(
            "PartNumber"=>2,
            "ETag"=>"7778aef83f66abc1fa1e8477f296d394"
            ),
        array(
            "PartNumber"=>3,
            "ETag"=>"7778aef83f66abc1fa1e8477f296d394"
            ),
        array(
            "PartNumber"=>4,
            "ETag"=>"7778aef83f66abc1fa1e8477f296d394"
            ),
    )
)
```

使用示例:

```
$client->completeMultipartUpload($args);
```

5.3.13.2 添加回调

参数格式:
在原有参数的基础上加上如下

```
"CallBack"=>array(
    "Url"=>"<KS3通知的URL>",
    "BodyMagicVariables"=>array("bucket"=>"bucket","key"=>"key"),//魔法变量，key=>value中的value将被最后的实际值替换，比如"bucket"=>"bucket"替换为"bucket"=>"<上传的bucket>"。支持:"bucket","key","etag","objectSize","mimeType","createTime"
    "BodyVariables"=>array("name"=>"lijunwei")//自定义KS3回调时需要在body中带的参数
)
```

5.3.14 Abort Multipart Upload

取消分块上传
参数格式:

```
$args=array(
    "Bucket"=>"<您的bucket名称>",
    "Key"=>"<key>",
    "Options"=>array("uploadId"=><uploadId>)//由Initiate Multipart Upload获得
    );
```

使用示例:

```
$client->abortMultipartUpload($args);
```

5.3.15 Mutipart Upload Demo

分块上传示例

```
function multipartUpload($client){
    //初始化分开上传，获取uploadid
    $args = array(
        "Bucket"=>"aaphp",
        "Key"=>"multi.zip",
        "UserMeta"=>array(
        "x-kss-meta-test"=>"example"
        ),
    "ObjectMeta"=>array(
        "Content-Type"=>"text/plain"
        )
    );
    $uploadid = $client->initMultipartUpload($args);
    print_r($uploadid);
    $uploadid = $uploadid["UploadId"];//获取到uploadid
    echo $uploadid."\r\n";
    //开始上传

    $file = "D://iToolsSetup_3.1.6.6.1419818705.exe";//要上传的文件
    $partsize = 1024*1024*5;
    $resource = fopen($file,"r");
    $stat = fstat($resource);
    $total = $stat["size"];//获取文件的总大小
    fclose($resource);
    $count = (int)(($total-1)/$partsize) +1;//计算文件需要分几块上传
    echo $count."\r\n";
    for($i = 0;$i < $count;$i++){
        //依次上传每一块
        echo "upload".$i."\r\n";
        $args=array(
            "Bucket"=>"aaphp",
            "Key"=>"multi.zip",
            "Options"=>array(
                "partNumber"=>$i+1,
                "uploadId"=>$uploadid
            ),
            "ObjectMeta"=>array(
                "Content-Length"=>min($partsize,$total-$partsize*$i)//每次上传$partsize大小
            ),
            "Content"=>array(
                "content"=>$file,
                "seek_position"=>$partsize*$i//跳过之前已经上传的
            )
        );
        $etag = $client->uploadPart($args);
        print_r($etag);
```

```
        $etag = $etag["ETag"];
    }
    $parts = $client->listParts(array("Bucket"=>"aaphp","Key"=>"multi.zip","Options"=>array("uploadId"=>$uploadid)));
    print_r($parts);//列出以及上传的块
    //结束上传
    $args=array(
        "Bucket"=>"aaphp",
        "Key"=>"multi.zip",
        "Options"=>array("uploadId"=>$uploadid),
        "Parts"=>$parts["Parts"]//使用之前列出的块完成分开上传
    );
    $result = $client->completeMultipartUpload($args);
    print_r($result);
}
```

5.3.16 PUT Object - rename

文件重命名 参数格式:

```
$args = array(
    "Bucket"=>" bucket",
    "Key"=>" 源key",
    "newKey"=>" 修改后key "
    );
```

使用示例:

```
$client->renameObject ($args);
```

5.3.17 使用外链操作

在使用外链时，不支持跨域访问。

使用示例：

```
$args=array(
    "Method"="GET",//http请求方法
    "Bucket"=>"<bucket>",
    "Key"=>"<key>",
    "Options"=>array(
        "Expires"=><过期时间,单位秒,即x秒后过期>
        //其他参数...
        ),
    "Headers"=>array(
        //"Content-Type"=>"...",
        //"Content-MD5"=>"...",
        //"x-kss-acl"=>"..."
        )
    );
```

使用示例：

```
$client->generatePresignedUrl($args);
```

5.4 客户端加密

5.4.1 环境准备

添加mcrypt与openssl拓展

5.4.2 初始化客户端

- 使用256位AES对称主密钥

  ```
  $client = new Ks3EncryptionClient("","","");
  ```

- 使用1024位RSA非对称主密钥(密钥对)

  ```
  $client = new Ks3EncryptionClient("","",array("",""));
  ```

5.4.3 注意事项

- 上传上去的文件是经过加密的。
- 下载文件只能通过该客户端getObject方法下载，用其他方法下载下来的文件是经过加密的。

- 分块上传时必须依次上传每一块。当上传最后一块时必须通过$args=array("LastPart"=>TRUE)指定最后一块。上传顺序不能错乱，不能使用多线程分块上传。
- 请妥善保管自己的主密钥，如果主密钥丢失，将无法解密数据。

iOS

KS3 SDK for iOS使用指南

SDK下载地址

- 点击下载

目录

开发前准备

SDK使用准备

- 申请AccessKey、SecretKey

SDK配置

SDK以动态库的形式呈现。请将*KS3iOSSDK.framework*添加到项目工程中。如果开发工具是Xcode6，请在*project->target->General*中的‘Embedded Binaries‘中添加*KS3iOSSDK.framework*。

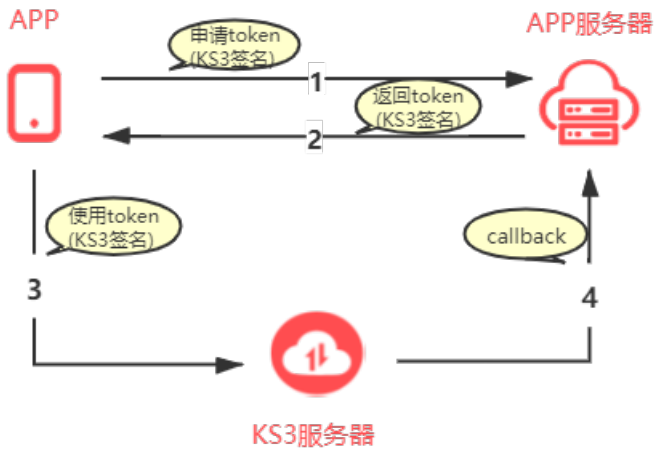
运行环境

支持iOS6及以上版本

安全性

使用场景

由于在App端明文存储AccessKey、SecretKey是极不安全的，因此推荐的使用场景如下图所示：



KS3Client初始化

- 利用AccessKey、SecretKey初始化

对应的初始化代码如下：

```
[[KS3Client initialize] connectWithAccessKey:strAccessKey withSecretKey:strSecretKey];
```

常见术语介绍

Object（对象，文件）

在KS3中，用户操作的基本数据单元是Object。单个Object允许存储0~48.8TB的数据。 Object 包含key和data。其中，key是Object的名字；data是Object 的数据。key为UTF-8编码，且编码后的长度不得超过1024个字符。

Key（文件名）

即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字符。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

其他术语请参考概念与术语

SDK介绍及使用

核心类介绍

- KS3Client 封装接入Web Service的一系列操作，提供更加便利的接口以及回调。

  为方便开发者使用，SDK在REST API接口返回值基础上进行了封装，具体更多封装类详情请见 SDK-REST API

资源管理操作

- List Bucket 列出客户所有的Bucket信息
- Create Bucket 创建一个新的Bucket
- Delete Bucket 删除指定Bucket
- Get Bucket ACL 获取Bucket的ACL
- Put Bucket ACL 设置Bucket的ACL
- Head Bucket 查询是否已经存在指定Bucket
- Get Object 下载Object数据
- Head Object 查询是否已经存在指定Object
- Delete Object 删除指定Object
- Get Object ACL 获得Bucket的acl
- Put Object ACL 上传object的acl
- List Objects 列举Bucket内的Object
- Put Object 上传Object数据
- Initiate Multipart Upload 调用这个接口会初始化一个分块上传
- Upload Part 上传分块
- List Parts 罗列出已经上传的块

- Abort Multipart Upload 取消分块上传
- Complete Multipart Upload 组装所有分块上传的文件
- Multipart Upload Example Code 分块上传代码示例
- Upload Manager 基于分块上传的简单上传接口

Service操作

List Bucket

列出客户所有的 Bucket 信息

方法名：

```
- (NSArray *)listBuckets:(KS3ListBucketsRequest *)request
```

参数说明：

- 无

返回结果：

- 客户所有的bucket列表，列表中每个元素是KS3Buckket对象

代码示例：

```
KS3ListBucketsRequest *request = [[KS3ListBucketsRequest alloc] init];
[request setBucket:@"bucket"];
NSArray *arrBuckets = [[KS3Client initialize] listBuckets:request];
```

Bucket操作

Creat Bucket

创建一个新的Bucket

方法名：

```
- (KS3CreateBucketResponse *)createBucket:(KS3CreateBucketRequest *)request
```

参数说明：

- bucketName：指定的Bucket名称

返回结果：

- 创建Bucket的HTTP请求响应

代码示例：

```
KS3CreateBucketRequest *request = [[KS3CreateBucketRequest alloc] initWithName:@"bucket"];
KS3CreateBucketResponse *response = [[KS3Client initialize] createBucket:request];
```

Delete Bucket

删除指定Bucket

方法名：

```
- (KS3DeleteBucketResponse *)deleteBucket:(KS3DeleteBucketRequest *)bucketName;
```

参数说明：

- bucketName ： 指定的Bucket名称

返回结果：

- 删除Bucket的HTTP请求响应

代码示例：

```
KS3DeleteBucketRequest *request = [[KS3DeleteBucketRequest alloc] initWithName:@"bucket"];
KS3DeleteBucketResponse *response = [[KS3Client initialize] deleteBucket:request];
```

Get Bucket ACL

获取Bucket的ACL

方法名：

- (KS3GetACLResponse *)getBucketACL:(KS3GetACLRequest *)getACLRequest

参数说明：

- getACLRequest：获取Bucket ACL的KS3GetACLRequest对象

返回结果：

- 获取Bucket ACL的HTTP请求响应

代码示例：

```
KS3GetACLRequest * getACLRequest = [[KS3GetACLRequest alloc] initWithName: @"blues111"];
KS3GetACLResponse * response = [[KS3Client initialize] getBucketACL: getACLRequest];
KS3BucketACLResult * result = response.listBucketsResult;
if (response.httpStatusCode == 200) {
    NSLog(@"Get bucket acl success!");

    NSLog(@"Bucket owner ID:          %@", result.owner.ID);
    NSLog(@"Bucket owner displayName: %@", result.owner.displayName);

    for (KS3Grant * grant in result.accessControlList) {
        NSLog(@"%@", grant.grantee.ID);
        NSLog(@"%@", grant.grantee.displayName);
        NSLog(@"%@", grant.grantee.URI);
        NSLog(@"%@", grant.permission);
    }
}
else {
    NSLog(@"Get bucket acl error: %@", response.error.description);
}
```

Put Bucket ACL

设置Bucket的ACL，以AccessControlList

方法名：

- (KS3SetGrantACLResponse *)setGrantACL:(KS3SetGrantACLRequest *)setGrantACLRequest;

参数说明：

- setGrantACLRequest：设置Bucket Grant ACL的KS3SetGrantACLRequest对象

返回结果：

- 设置Bucket Grant ACL的HTTP请求响应

代码示例：

```
KS3GrantAccessControlList * acl = [[KS3GrantAccessControlList alloc] init];
//设置权限为公共读
[acl setGrantControlAccess: KingSoftYun_Grant_Permission_Read];
//设置被授权人的id和name
acl.identifier = @"523678123";
acl.displayName = @"blues111";
KS3SetGrantACLRequest * request = [[KS3SetGrantACLRequest alloc] initWithName: @""accessACL: acl];
KS3SetGrantACLResponse * response = [[KS3Client initialize] setGrantACL: request];
if (response.httpStatusCode == 200) {
    NSLog(@"Set grant acl success!");
}
else {
    NSLog(@"Set grant acl error: %@", response.error.description);
}
```

Head Bucket

查询是否已经存在指定Bucket

方法名：

- (KS3HeadBucketResponse *)headBucket:(KS3HeadBucketRequest *)headBucketRequest

参数说明：

- headBucketRequest:查询是否已存在指定的Bucket的KS3HeadBucketRequest请求

返回结果：

- 查询是否已存在指定的Bucket的HTTP请求响应

代码示例：

```
KS3HeadBucketRequest * request = [[KS3HeadBucketRequest alloc] initWithName: @"blues111"];
KS3HeadBucketResponse * response = [[KS3Client initialize] headBucket: request];
if (response.httpStatusCode == 200) {
    NSLog(@"Head bucket success!");
}
else {
    NSLog(@"Head bucket error: %@", response.error.description);
}
```

Object操作

Get Object

下载该Object数据

方法名：

- (KS3DownLoad *)downloadObjectWithBucketName:(NSString *)strBucketName key:(NSString *)strObject downloadBeginBlock:(KS3DownloadBeginBlock)downloadBeginBlock downloadFileCompleteion:(KS3DownloadFileCompleteionBlock)downloadFileCompleteion downloadProgressChangeBlock:(KS3DownloadProgressChangeBlock)downloadProgressChangeBlock failedBlock:(KS3DownloadFailedBlock)failedBlock;

参数说明：

- strBucketName：指定的Bucket名称
- strObjName：指定的Object名称
- downloadBeginBlock：表示下载开始的block
- downloadFileCompleteion：表示下载完成后的block
- downloadProgressChangeBlock:表示下载中的block
- failedBlock：表示错误处理的block

返回结果：

- 下载Object的KS3DownLoad对象

代码示例：

```
[[KS3Client initialize] downloadObjectWithBucketName: @"photo_hor.jpeg"key: @"alert1"downloadBeginBlock: ^(KS3DownLoad * aDownload, NSURLResponse * responseHeaders) {

}downloadFileCompleteion: ^(KS3DownLoad * aDownload, NSString * filePath) {

}downloadProgressChangeBlock: ^(KS3DownLoad * aDownload, double newProgress) {

}failedBlock: ^(KS3DownLoad * aDownload, NSError * error) {

}];
```

Head Object

查询是否已经存在指定Object

方法名：

- (KS3HeadObjectResponse *)headObject:(KS3HeadObjectRequest *)headObjectRequest

参数说明：

- headObjectRequest：查询Object是否存在的KS3HeadObjectRequest对象

返回结果：

- 查询指定Object是否存在的HTTP请求响应

代码示例：

```
KS3HeadObjectRequest * headObjRequest = [[KS3HeadObjectRequest alloc] initWithName: strBucketName withKeyName: strObjectName];
KS3HeadObjectResponse * response = [[KS3Client initialize] headObject: headObjRequest];
if (response.httpStatusCode == 200) {
    NSLog(@"Head object success!");
}
else {
    NSLog(@"Head object error: %@", response.error.description);
}
```

Delete Object

删除指定Object

方法名：

```
- (KS3DeleteObjectResponse *)deleteObject:(KS3DeleteObjectRequest *)deleteObjectRequest;
```

参数说明：

- deleteObjectRequest：删除Object的KS3DeleteObjectRequest对象

返回结果：

- 删除指定Object是否存在的HTTP请求响应

代码示例：

```
KS3DeleteObjectRequest * deleteObjRequest = [[KS3DeleteObjectRequest alloc] initWithName: strBucketName withKeyName: strObjectNam
e];
KS3DeleteObjectResponse * response = [[KS3Client initialize] deleteObject: deleteObjRequest];
if (response.httpStatusCode == 200) {
    NSLog(@"Delete object success!");
}
else {
    NSLog(@"Delete object error: %@", response.error.description);
}
```

Get Object ACL

获得Object的acl

方法名：

```
- (KS3GetObjectACLResponse *)getObjectACL:(KS3GetObjectACLRequest *)getObjectACLRequest;
```

参数说明：

- getObjectACLRequest：获取Object ACL的KS3GetObjectACLRequest对象

返回结果：

- 获取Object ACL的HTTP请求响应

代码示例：

```
KS3GetObjectACLRequest * getObjectACLRequest = [[KS3GetObjectACLRequest alloc] initWithName: strBucketName withKeyName: strObjectN
ame];
KS3GetObjectACLResponse * response = [[KS3Client initialize] getObjectACL: getObjectACLRequest];
KS3BucketACLResult * result = response.listBucketsResult;
if (response.httpStatusCode == 200) {

    NSLog(@"Object owner ID:          %@", result.owner.ID);
    NSLog(@"Object owner displayName: %@", result.owner.displayName);

    for (KS3Grant * grant in result.accessControlList) {
        NSLog(@"%@", grant.grantee.ID);
        NSLog(@"%@", grant.grantee.displayName);
        NSLog(@"%@", grant.grantee.URI);
        NSLog(@"%@", grant.permission);
    }
}
else {
    NSLog(@"Get object acl error: %@", response.error.description);
```

```
}
```

Put Object ACL

上传object的acl，以CannedAccessControlList形式

方法名：

```
- (KS3SetObjectACLResponse *)setObjectACL:(KS3SetObjectACLRequest *)setObjectACLRequest;
```

参数说明：

- setObjectACLRequest：设置Object ACL的KS3SetObjectACLRequest对象

返回结果：

- 获取Object ACL的HTTP请求响应

代码示例：

```
KS3AccessControlList * acl = [[KS3AccessControlList alloc] init];
//设置object的ACL为私有
[acl setContronAccess: KingSoftYun_Permission_Private];
KS3SetObjectACLRequest * setObjectACLRequest = [[KS3SetObjectACLRequest alloc] initWithName: strBucketName withKeyName: strObjectN
ame acl: acl];
KS3SetObjectACLResponse * response = [[KS3Client initialize] setObjectACL: setObjectACLRequest];
if (response.httpStatusCode == 200) {
    NSLog(@"Set object acl success!");
}
else {
    NSLog(@"Set object acl error: %@", response.error.description);
}
```

上传object的acl，以AccessControlList形式

方法名：

```
- (KS3SetObjectGrantACLResponse *)setObjectGrantACL:(KS3SetObjectGrantACLRequest *)setObjectGrantACLRequest;
```

参数说明：

- setObjectGrantACLRequest：设置Object Grant ACL的KS3SetObjectGrantACLRequest对象

返回结果：

- 获取Object Grant ACL的HTTP请求响应

代码示例：

```
KS3GrantAccessControlList * acl = [[KS3GrantAccessControlList alloc] init];
//设置权限为公共读
[acl setGrantControlAccess: KingSoftYun_Grant_Permission_Read];
//设置被授权人的id和name
acl.identifier = @"436749834";
acl.displayName = @"blues111";
KS3SetObjectGrantACLRequest * request = [[KS3SetObjectGrantACLRequest alloc] initWithName: @"blues111"withKeyName: @"500.txt"grant
Acl: acl];
KS3SetObjectGrantACLResponse * response = [[KS3Client initialize] setObjectGrantACL: request];
if (response.httpStatusCode == 200) {
    NSLog(@"Set object grant acl success!");
}
else {
    NSLog(@"Set object grant acl error: %@", response.error.description);
}
```

List-Objects

列举Bucket内的Object

方法名：

```
- (KS3ListObjectsResponse *)listObjects:(KS3ListObjectsRequest *)listObjectsRequest;
```

参数说明：

- listObjectsRequest：列举指定的Bucket内所有Object的KS3ListObjectsRequest对象，它可以设置

prefix，marker，maxKeys，delimiter四个指定的属性。prefix：限定返回的Object名字都以制定的prefix前缀开始。类型：字符串默认：无；marker：从一个指定的名字marker开始列出Object的名字。类型：字符串默认值：无；maxKeys：设定返回的Object名字数量，返回的数量有可能比设定的少，但是绝不会比设定的多，如果还存在没有返回的Object名字，返回的结果包含<IsTruncated>true</IsTruncated>。类型：字符串默认：10000；delimiter：delimiter是用来对Object名字进行分组的一个字符。包含指定的前缀到第一次出现的delimiter字符的所有Object名字作为一组结果CommonPrefix。类型：字符串默认值：无

返回结果：

- 列举指定Bucket内所有Object的HTTP请求响应

代码示例：

```
KS3ListObjectsRequest * listObjectRequest = [[KS3ListObjectsRequest alloc] initWithName: @"blues111"];
KS3ListObjectsResponse * response = [[KS3Client initialize] listObjects: listObjectRequest];
KS3ListObjectsResult * _result = response.listBucketsResult;
NSMutableArray * _arrObjects = response.listBucketsResult.objectSummaries;

for (KS3ObjectSummary * objectSummary in _arrObjects) {
    NSLog(@"%@", objectSummary.Key);
    NSLog(@"%@", objectSummary.owner.ID);
}
NSLog(@"%@", _result.bucketName);
NSLog(@"%ld", _result.objectSummaries.count);
NSLog(@"%ld", _result.commonPrefixes.count);

NSLog(@"KS3ListObjectsResponse %d", response.httpStatusCode);
```

Put-Object

上传Object数据

方法名：

```
- (KS3PutObjectResponse *)putObject:(KS3PutObjectRequest *)putObjectRequest;
```

参数说明：

- putObjectRequest：上传指定的Object的KS3PutObjectRequest对象。它需要设置指定的Bucket的名称和Object的名称

返回结果：

- 上传指定的Object的HTTP请求响应

代码示例：

1.普通上传

```
/* 一定要实现委托方法（这种情况如果实现委托，返回的reponse一般返回为nil，具体获取返回对象需要到委托方法里面获取，如果不实现委托，repon
se不会为nil*/
KS3PutObjectRequest *putObjRequest = [[KS3PutObjectRequest alloc] initWithName:@"testcreatebucket-wf111" withAcl:nil grantAcl:nil]
;
//设置object权限为公开读
KS3AccessControlList *acl = [[KS3AccessControlList alloc] init];
[acl setContronAccess:KingSoftYun_Permission_Public_Read];
[putObjRequest setAcl:acl];
putObjRequest.delegate = self;
NSString *fileName = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"jpg"];
putObjRequest.data = [NSData dataWithContentsOfFile:fileName options:NSDataReadingMappedIfSafe error:nil];
putObjRequest.filename = [fileName lastPathComponent];

[[KS3Client initialize] putObject:putObjRequest];
```

2.有些情况下，如果希望KS3上传文件完成后通知服务端，需要注册回调参数。使用方法：

```
/* 一定要实现委托方法（这种情况如果实现委托，返回的reponse一般返回为nil，具体获取返回对象需要到委托方法里面获取，如果不实现委托，repon
se不会为nil*/
KS3PutObjectRequest *putObjRequest = [[KS3PutObjectRequest alloc] initWithName:@"testcreatebucket-wf111" withAcl:nil grantAcl:nil]
;
//设置object权限为公开读
KS3AccessControlList *acl = [[KS3AccessControlList alloc] init];
[acl setContronAccess:KingSoftYun_Permission_Public_Read];
[putObjRequest setAcl:acl];
putObjRequest.delegate = self;
NSString *fileName = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"jpg"];
putObjRequest.data = [NSData dataWithContentsOfFile:fileName options:NSDataReadingMappedIfSafe error:nil];
putObjRequest.filename = [fileName lastPathComponent];
//设置回调函数
```

```
putObjRequest.callbackUrl = @"http://123.59.36.81/index.php/api/photos/callback";
//设置回调参数，回调参数支持自定义参数、常量和魔法变量。如下所示：kss-location和kss-name为自定义参数，key和etag为魔法变量。
//支持的魔法变量详见上传回调处理说明文档([https://docs.ksyun.com/documents/956](https://docs.ksyun.com/documents/956))
putObjRequest.callbackBody = @"location=${kss-location}&name=${kss-name}&uid=8888&objectKey=${key}&etag=${etag}";
//设置自定义参数，必须以"kss-"开始
putObjRequest.callbackParams = @{@"kss-location": @"china_location", @"kss-name": @"lulu_name"};

[[KS3Client initialize] putObject:putObjRequest];
```

Initiate Multipart Upload

调用这个接口会初始化一个分块上传，KS3 Server会返回一个upload id, upload id 用来标识属于当前object的具体的块，并且用来标识完成分块上传或者取消分块上传

方法名：

```
- (KS3MultipartUpload *)initiateMultipartUploadWithKey:(NSString *)theKey withBucket:(NSString *)theBucket
```

参数说明：

- theKey：指定的Object名称
- theBucket：指定的Bucket名称

返回结果：

- 初始化分块上传的HTTP响应，KS3MultipartUpload类型的对象里面包含了指定的Object名称，Bucket名称，此次上传的Upload ID，Object的Owner，初始化日期

代码示例：

```
KS3InitiateMultipartUploadRequest *request = [[KS3InitiateMultipartUploadRequest alloc] initWithKey:strObjectName inBucket:strBuck
etName acl:nil grantAcl:nil];
KS3MultipartUpload *upload = [[KS3Client initialize] initiateMultipartUploadWithRequest:request];
```

Upload Part

初始化分块上传后，上传分块接口。Part number 是标识每个分块的数字，介于0-10000之间。除了最后一块，每个块必须大于等于5MB，最后一块没有这个限制。

方法名：

```
- (KS3UploadPartResponse *)uploadPart:(KS3UploadPartRequest *)uploadPartRequest;
```

参数说明：

- uploadPartRequest：上传块的KS3UploadPartRequest对象，它需要指定上传的Object名称，指定的Bucket的名称，分块的块号，此块的数据

返回结果：

- 块上传的HTTP请求响应

代码示例：

```
KS3UploadPartRequest *req = [[KS3UploadPartRequest alloc] initWithMultipartUpload:upload partNumber:partNumber data:data generate
MD5:NO];
req.delegate = self;
KS3UploadPartResponse *response = [[KS3Client initialize] uploadPart:req];
```

List Parts

罗列出已经上传的块

方法名：

```
- (KS3ListPartsResponse *)listParts:(KS3ListPartsRequest *)listPartsRequest;
```

参数说明：

- listPartsRequest：罗列已经上传的块的KS3ListPartsRequest对象，它包含指定的Object的名称，此次上传的Upload ID，maxParts，它表示块大小限制，类型：字符串，默认值：None，partNumberMarker，它表示块号标记，将返回大于此块号的分块，类型：字符串，默认值：None

返回结果：

- 罗列已经上传的块的HTTP请求响应，它包含了类型为KS3ListPartsResult的请求的结果，它包含指定的Bucket名称，指定的Object名称，此次上传的Upload ID，partNumberMarker，它表示块号标记，将返回大于此块号的分块，maxParts，它表示块大小限制，isTruncated，它表示是否取完分块，Owner它表示创建分块上传的用户

代码示例：

```
KS3ListPartsRequest *req2 = [[KS3ListPartsRequest alloc] initWithMultipartUpload:_muilt];
KS3ListPartsResponse *response2 = [[KS3Client initialize] listParts:req2];
```

Abort Multipart Upload

取消分块上传。

方法名：

```
- (KS3AbortMultipartUploadResponse *)abortMultipartUpload:(KS3AbortMultipartUploadRequest
*)abortMultipartRequest
```

参数说明：

- abortMultipartRequest：取消分块上传的KS3AbortMultipartUploadRequest对象，它需要使用KS3MultipartUpload对象来初始化，初始化包括指定的Bucket名称，Object的名称，分块上传的Upload ID

返回结果：

- 取消上传的HTTP请求响应

代码示例：

```
KS3AbortMultipartUploadRequest * request = [[KS3AbortMultipartUploadRequest alloc] initWithMultipartUpload: _muilt];
KS3AbortMultipartUploadResponse * response = [[KS3Client initialize] abortMultipartUpload: request];
if (response.httpStatusCode == 204) {
    NSLog(@"Abort multipart upload success!");
}
else {
    NSLog(@"error: %@", response.error.description);
}
```

Complete Multipart Upload

组装之前上传的块，然后完成分块上传。通过你提供的xml文件，进行分块组装。在xml文件中，块号必须使用升序排列。必须提供每个块的ETag值。

方法名：

```
- (KS3CompleteMultipartUploadResponse *)completeMultipartUpload:(KS3CompleteMultipartUploadRequest
*)completeMultipartUploadRequest;
```

参数说明：

- completeMultipartUploadRequest：组装上传所有块的KS3CompleteMultipartUploadRequest对象，它包含指定的Bucket名称，Object名称，此次上传的Upload ID，需要组装的所有块的信息数据

返回结果：

- 组装所有块的HTTP请求响应

代码示例：

```
KS3ListPartsResponse * response2 = [[KS3Client initialize] listParts: req2];
KS3CompleteMultipartUploadRequest * req = [[KS3CompleteMultipartUploadRequest alloc] initWithMultipartUpload: _muilt];
for (KS3Part * part in response2.listResult.parts) {
    [req addPartWithPartNumber: part.partNumber withETag: part.etag];
}
[[KS3Client initialize] completeMultipartUpload: req];
```

Multipart Upload Example Code

分块上传代码示例

```
NSFileHandle * fileHandle = [NSFileHandle fileHandleForReadingAtPath: [[NSBundle mainBundle] pathForResource: @"bugDownload"ofTy
pe: @"txt"]];
```

```
long long fileLength = [[fileHandle availableData] length];
long long partLength = 5 * 1024.0 * 1024.0;
_partInter = (ceilf((float) fileLength / (float) partLength)); [fileHandle seekToFileOffset: 0];

KS3InitiateMultipartUploadRequest * _muilt = [[KS3Client initialize] initiateMultipartUploadWithKey: @"500.txt"withBucket: @"blues
111"inBucket: strBucketName acl: nil grantAcl: nil];
for (NSInteger i = 0; i < _partInter; i++) {
    NSData * data = nil;
    if (i == _partInter - 1) {
        data = [fileHandle readDataToEndOfFile];
    }
    else {
        data = [fileHandle readDataOfLength: partLength];
        [fileHandle seekToFileOffset: partLength * (i + 1)];
    }
    KS3UploadPartRequest * req = [[KS3UploadPartRequest alloc] initWithMultipartUpload: _muilt];
    req.delegate = self;
    req.data = data;
    req.partNumber = (int32_t) i + 1;
    req.contentLength = data.length; [[KS3Client initialize] uploadPart: req];
}

// **** 分块上传的回调，每块上传结束后都会被调用
- (void) request: (KS3ServiceRequest * ) request didCompleteWithResponse: (KS3ServiceResponse * ) response {
    _upLoadCount++;
    if (_partInter == _upLoadCount) {
        KS3ListPartsRequest * req2 = [[KS3ListPartsRequest alloc] initWithMultipartUpload: _muilt];
        KS3ListPartsResponse * response2 = [[KS3Client initialize] listParts: req2];
        KS3CompleteMultipartUploadRequest * req = [[KS3CompleteMultipartUploadRequest alloc] initWithMultipartUpload: _muilt];
        NSLog(@" - - - - - %@", response2.listResult.parts);
        for (KS3Part * part in response2.listResult.parts) { [req addPartWithPartNumber: part.partNumber withETag: part.etag];
        }
        [[KS3Client initialize] completeMultipartUpload: req];
    }
}
- (void) request: (KS3ServiceRequest * ) request didFailWithError: (NSError * ) error {
    NSLog(@"error: %@", error.description);
}
```

Upload Manager

分块上传接口可以实现断点续传等功能，但是由于使用比较复杂，SDK提供了一个封装后的接口——Ks3UploadManager。

使用本接口首先需要初始化一个KS3UploadManager实例。

```
self.uploadManager = [KS3UploadManager sharedInstanceWithClient:[KS3Client initialize] authHandler:nil];
```

KS3UploadManager接受唯一参数是authHandler，此handler用于处理鉴权串签名，每次请求都会用本次请求的信息调用authHandler，handler方法内，请求鉴权服务器拿到签名串返回即可。

authHandler如果为nil，则需要在客户端设置AK、SK（这种方式不推荐）。设置方法为：[[KS3Client initialize] setCredentials:[[KS3Credentials alloc] initWithAccessKey:"YOUR_KS3_ACCESS_KEY" withSecretKey:"YOUR_KS3_SECRET_KEY"]];。

实例创建好后，可以调用上传方法：

```
// 读取文件信息
NSString * strFilePath = [[NSBundle mainBundle] pathForResource: @"7.6M"ofType: @"mov"];
NSData * data = [NSData dataWithContentsOfFile: strFilePath];

KS3AccessControlList * acl = [[KS3AccessControlList alloc] init]; [acl setContronAccess: KingSoftYun_Permission_Public_Read];

// 创建一次开始分块的请求
KS3UploadRequest * uploadRequest = [[KS3UploadRequest alloc] initWithKey: @"uploadmanager/sample.mov"inBucket: kUploadBucketName
acl: acl grantAcl: nil]; [uploadRequest setCompleteRequest]; [uploadRequest setStrKS3Token: [KS3Util getAuthorization: uploadReque
st]];

// 开始上传
[self.uploadManager putData: data request: uploadRequest blockSize: 1 * kMB progress: ^(NSString * key, double percent) {
    NSLog(@"objectKey: %@, progress %lf", key, percent);
}
cancelSignal: ^BOOL(NSString * key) {
    return false; // 修改这里进行取消
}
complete: ^(KS3Upload * upload, KS3Response * response) {
    NSLog(@"uploadId: %@, response %@", upload.uploadId, response);
}
error: ^(KS3Upload * upload, NSError * error) {
    NSLog(@"uploadId: %@, error: %@", upload.uploadId, error);
}];
```

目前此接口只支持使用NSData作为参数进行上传。

其它

完整示例，请见 KS3-iOS-SDK-Demo

C

KS3 SDK For C# 使用指南

目录

---

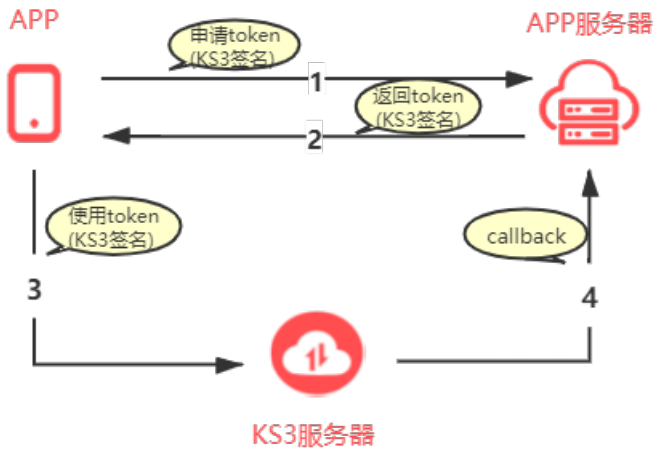
1. 概述

此SDK适用于.net framework4.0及以上版本。基于KS3 API 构建。使用此 SDK 构建您的网络应用程序，能让您以非常便捷地方式将数据安全地存储到金山云存储上。无论您的网络应用是一个网站程序，还是包括从云端（服务端程序）到终端（手持设备应用）的架构的服务或应用，通过KS3存储及其 SDK，都能让您应用程序的终端用户高速上传和下载，同时也让您的服务端更加轻盈。

2. 环境准备

- 配置.net framework4.0 以上开发环境
- 点击下载KS3 SDK

由于在App端明文存储AccessKey、SecretKey是极不安全的，因此推荐的使用场景如下图所示：



3. 初始化

3.1 获取密钥

1. 开通KS3服务，https://www.ksyun.com/user/register 注册账号

2. 进入控制台，https://ks3.console.ksyun.com/console.html#/setting 获取AccessKeyID 、AccessKeySecret

3.2 初始化客户端

当以上全部完成之后用户便可初始化客户端进行操作

```
String accessKey = "YOUR ACCESS KEY";
String secretKey = "YOUR SECRET KEY";
String bucketName = "YOUR BUCKET NAME";
String endPoint = "ks3-cn-beijing.ksyuncs.com"; //此处以北京region为例
ks3Client = new KS3Client(accessKey, secretKey);
ks3Client.setEndpoint(endPoint);
```

- endpoint与Region对应关系

3.3 常用术语介绍

Object（对象，文件）

在KS3中，用户操作的基本数据单元是Object。单个Object允许存储0~48.8TB的数据。 Object 包含key和data。其中，key是Object的名字；data是Object 的数据。key为UTF-8编码，且编码后的长度不得超过1024个字符。

Key（文件名）

即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字符。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

其他术语请参考概念与术语

4. 使用示例

4.1 List Buckets

使用示例

列出当前用户的所有bucket,可以查看每个bucket的名称、创建时间以及所有者

```
private static bool listBuckets()
{
    // List Buckets
    try
    {
        IList < Bucket > bucketsList = ks3Client.listBuckets();
        foreach(Bucket b in bucketsList)
```

```
        {
            Console.WriteLine(b.ToString());
        }
    }
    catch(System.Exception e)
    {
        return false;
    }
    return true;
}
```

4.2 DELETE Bucket

使用示例

删除一个Bucket

```
private static bool deleteBucket()
{
    // Delete Bucket
    try
    {
        ks3Client.deleteBucket(bucketName);
    }
    catch(System.Exception e)
    {
        return false;
    }
    return true;
}
```

4.3 List Objects

使用示例

列出<bucket名称>下的所有object，最大上限是1000个

```
private static bool listObjects()
{
    try
    {
        // List Objects
        Console.WriteLine("--- List Objects: ---");
        ObjectListing objects = ks3Client.listObjects(bucketName);
        Console.WriteLine(objects.ToString());
        Console.WriteLine("----------------------\n");

        // Get Object Metadata
        Console.WriteLine("--- Get Object Metadata ---");
        ObjectMetadata objMeta = ks3Client.getObjectMetadata(bucketName, objKeyName);
        Console.WriteLine(objMeta.ToString());
        Console.WriteLine("----------------------------\n");
    }
    catch(System.Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.4 GET Bucket acl

使用示例

获取<bucket名称>的acl控制权限

```
private static bool getBucketACL()
{
    // 获取 Bucket ACL
    try
    {
        Console.WriteLine("--- Get Bucket ACL: ---");
        AccessControlList acl = ks3Client.getBucketAcl(bucketName);
        Console.WriteLine("Bucket Name: " + bucketName);
        Console.WriteLine(acl.ToString());
        Console.WriteLine("-----------------------\n");
    }
```

```
    catch(System.Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.5 List Multipart Uploads

使用示例

列出当前正在执行的分块上传

```
private static ListMultipartUploadsResult listMultipartUploads(string bucket, string objKey, string uploadId) {
    ListMultipartUploadsRequest request = new ListMultipartUploadsRequest(bucket, objKey, uploadId);
    ListMultipartUploadsResult result = ks3Client.getListMultipartUploads(request);
    return result;
}
```

4.6 Create Bucket

使用示例

新建一个<bucket名称>

```
private static bool createBucket()
{
    // Create Bucket
    try
    {
        Console.WriteLine("--- Create Bucket: ---");
        Console.WriteLine("Bucket Name: " + bucketName);
        Bucket bucket = ks3Client.createBucket(bucketName);
        Console.WriteLine("Success.");
        Console.WriteLine("----------------------\n");
    }
    catch(System.Exception e)
    {
        Console.WriteLine("Create Bucket Fail! " + e.ToString());
        return false;
    }
    return true;
}
```

注：这里如果出现409 conflict错误，说明请求的bucket name有冲突，因为bucket name是全局唯一的

4.7 PUT Bucket acl

使用示例

设置<bucket名称>的访问权限

```
private static bool setBucketACL()
{
    // 为bucket设置公共读写权限
    try
    {
        Console.WriteLine("--- Set Bucket ACL: ---");
        CannedAccessControlList cannedAcl = new CannedAccessControlList(CannedAccessControlList.PUBLICK_READ_WRITE);
        ks3Client.setBucketAcl(bucketName, cannedAcl);
        Console.WriteLine("Bucket Name: " + bucketName);
        Console.WriteLine("Success, now the ACL is:\n" + ks3Client.getBucketAcl(bucketName));
        Console.WriteLine("------------------------\n");
    }
    catch(System.Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.8 DELETE Object

使用示例

删除<bucket名称>内一个object

```
private static bool deleteObject()
{
    // Delete Object
    try
    {
        Console.WriteLine("--- Delete Object: ---");
        ks3Client.deleteObject(bucketName, objKeyName);
        Console.WriteLine("Delete Object completed.");
        Console.WriteLine("----------------------\n");
    }
    catch(Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.9 GET Object

使用示例

下载<bucket名称>下的object

```
private static bool getObject() {
    try {
        // GET Object为用户提供了object的下载，用户可以通过控制Range实现分块多线程下载
        Console.WriteLine("--- Download and Store in Memory ---");
        GetObjectRequest getShortContent = new GetObjectRequest(bucketName, objKeyName);
        getShortContent.setRange(0, 24);
        KS3Object ks3Object = ks3Client.getObject(getShortContent);
        StreamReader sr = new StreamReader(ks3Object.getObjectContent());
        Console.WriteLine("Content:\n" + sr.ReadToEnd());
        sr.Close();
        ks3Object.getObjectContent().Close();
        Console.WriteLine("-------------------------------------\n");
    } catch(System.Exception e) {
        Console.WriteLine(e.ToString());
        return false;
    }
    try {
        // 直接下载并存储成文件
        Console.WriteLine("--- Download a File ---");
        // I need to get the Content-Length to set the listener.
        ObjectMetadata objectMetadata = ks3Client.getObjectMetadata(bucketName, objKeyName);
        SampleListener downloadListener = new SampleListener(objectMetadata.getContentLength());
        GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objKeyName, new FileInfo(outFilePath));
        getObjectRequest.setProgressListener(downloadListener);
        KS3Object obj = ks3Client.getObject(getObjectRequest);
        obj.getObjectContent().Close(); // The file was opened in [KS3ObjectResponseHandler], so I close it first.
        Console.WriteLine("Success. See the file downloaded at {0}", outFilePath);
        Console.WriteLine("------------------------\n");
    } catch(System.Exception e) {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.10 GET Object acl

使用示例

获取 <bucket名称>这个bucket下<object key>的权限控制信息

```
private static bool getObjectACL()
{
    //获取 Object ACL
    try
    {
        Console.WriteLine("--- Get Object ACL: ---");
        AccessControlList acl = ks3Client.getObjectAcl(bucketName, objKeyName);
        Console.WriteLine("Object Key: " + objKeyName);
        Console.WriteLine(acl.ToString());
        Console.WriteLine("------------------------\n");
    }
    catch(System.Exception e)
    {
```

```
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.11 PUT Object

使用示例

将new File("<filePath>")这个文件上传至<bucket名称>这个存储空间下，并命名为<object key>

```
private static bool putObject()
{
    try
    {
        //Put Object(upload a short content)
        //以流方式上传文件到KS3，Content-Type 默认为 application/octet-stream
        Console.WriteLine("--- Upload a Short Content: ---");
        String sampleContent = "This is a sample content.(25 characters before, included the 4 spaces)";
        Stream stream = new MemoryStream(Encoding.UTF8.GetBytes(sampleContent));
        PutObjectResult shortContentResult = ks3Client.putObject(bucketName, objKeyName, stream, null);
        Console.WriteLine("Upload Completed. eTag=" + shortContentResult.getETag() + ", MD5=" + shortContentResult.getContentMD5()
);
        Console.WriteLine("-------------------------------\n");

        //Put Object(upload a file)
        //以文件方式上传到KS3时，Content-Type 会默认匹配上传文件的后缀名
        Console.WriteLine("--- Upload a File ---");
        FileInfo file = new FileInfo("E:\\tool\\eclipse.rar");
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, objKeyName, file);
        SampleListener sampleListener = new SampleListener(file.Length);
        putObjectRequest.setProgressListener(sampleListener);
        PutObjectResult putObjectResult = ks3Client.putObject(putObjectRequest);
        Console.WriteLine("Upload Completed. eTag=" + putObjectResult.getETag() + ", MD5=" + putObjectResult.getContentMD5());
        Console.WriteLine("----------------------\n");
    }
    catch(System.Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.12 PUT Object acl

使用示例

修改<bucket名称>下object的权限控制

```
private static bool setObjectACL()
{
    // 设置Object ACL为公共读写
    try
    {
        Console.WriteLine("--- Set Object ACL: ---");
        CannedAccessControlList cannedAcl = new CannedAccessControlList(CannedAccessControlList.PUBLICK_READ_WRITE);
        Console.WriteLine("Object Key: " + objKeyName);
        ks3Client.setObjectAcl(bucketName, objKeyName, cannedAcl);
        Console.WriteLine("Success, now the ACL is:\n" + ks3Client.getObjectAcl(bucketName, objKeyName));
        Console.WriteLine("------------------------\n");
    }
    catch(System.Exception e)
    {
        Console.WriteLine(e.ToString());
        return false;
    }
    return true;
}
```

4.13 Multipart Upload

使用示例

```
/**
     * 初始化分块上传，服务端会返回一个全局唯一的uploadid
     * **/
    private static InitiateMultipartUploadResult multipartUp()
```

```
    {
        InitiateMultipartUploadResult re=ks3Client.initiateMultipartUpload(bucketName, objKeyName);
        Console.WriteLine(re.ToString());
        return re;
    }
    /**
     * 分块上传例子
     * **/
    private static bool uploadPart() {
        string path = @"you file path";//上传文件路径,例如E:\tool\aa.rar
        InitiateMultipartUploadResult result=multipartUp();
        FileInfo file = new FileInfo(path);
        int part = 5 * 1024 * 1024;
        int numBytesToRead = (int)file.Length;
        int i = 0;
        XElement root = new XElement("CompleteMultipartUpload");//初始化一个xml，以备分块上传完成后调用complete方法提交本次上传的文件以通知服务端合并分块
        //开始读取文件
        using (FileStream fs = new FileStream(path, FileMode.Open))
        {
            while (numBytesToRead > 0)
            {
                UploadPartRequest request = new UploadPartRequest(
                        result.getBucket(), result.getKey(), result.getUploadId(),
                        i + 1);
                //每次读取5M文件内容，如果最后一次内容不及5M则按实际大小取值
                int count = Convert.ToInt32((i * part + part) > file.Length ? file.Length - i * part : part);
                byte[] data = new byte[count];
                int n = fs.Read(data, 0, count);
                request.setInputStream(new MemoryStream(data));
                ProgressListener sampleListener = new SampleListener(count);//实例一个更新进度的监听类，实际使用中可自己定义实现
                request.setProgressListener(sampleListener);
                PartETag tag = ks3Client.uploadPart(request);//上传本次分块内容
                Console.WriteLine(tag.ToString());
                if (n == 0)
                    break;
                numBytesToRead -= n;

                XElement partE = new XElement("Part");
                partE.Add(new XElement("PartNumber", i + 1));
                partE.Add(new XElement("ETag", tag.geteTag()));
                root.Add(partE);
                i++;
            }
        }
        //所有分块上传完成后发起complete request，通知服务端合并分块
        CompleteMultipartUploadRequest completeRequest = new CompleteMultipartUploadRequest(result.getBucket(), result.getKey(), result.getUploadId());
        completeRequest.setContent(new MemoryStream(System.Text.Encoding.Default.GetBytes(root.ToString())));
        CompleteMultipartUploadResult completeResult = ks3Client.completeMultipartUpload(completeRequest);
        return true;
    }

    /**
     * 放弃本次上传
     * **/
    private static bool AbortMultipartUpload(string bucket, string objKey, string uploadId)
    {
        AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(bucket, objKey, uploadId);
        ks3Client.AbortMultipartUpload(request);
        return true;
    }
/**
    * 多线程分块上传例子开始
    * **/
    static List<ManualResetEvent> manualEvents = new List<ManualResetEvent>();
    static List<Param> paras = new List<Param>();

    private static bool uploadPartMultithread()
    {
        string path = inFilePath;//上传文件路径,例如E:\tool\aa.rar
        InitiateMultipartUploadResult result = multipartUp();
        FileInfo file = new FileInfo(path);
        int part = 5 * 1024 * 1024;
        int numBytesToRead = (int)file.Length;
        int i = 0;
        XElement root = new XElement("CompleteMultipartUpload");//初始化一个xml，以备分块上传完成后调用complete方法提交本次上传的文件以通知服务端合并分块
        //开始读取文件
        using (FileStream fs = new FileStream(path, FileMode.Open))
        {
            while (numBytesToRead > 0)
            {
                UploadPartRequest request = new UploadPartRequest(
```

```
                        result.getBucket(), result.getKey(), result.getUploadId(),
                        i + 1);
                //每次读取5M文件内容，如果最后一次内容不及5M则按实际大小取值
                int count = Convert.ToInt32((i * part + part) > file.Length ? file.Length - i * part : part);
                byte[] data = new byte[count];
                int n = fs.Read(data, 0, count);
                request.setInputStream(new MemoryStream(data));

                ManualResetEvent mre = new ManualResetEvent(false);
                manualEvents.Add(mre);

                Param pra = new Param();
                pra.mrEvent = mre;
                pra.praNum = i + 1;
                pra.request = request;
                paras.Add(pra);

                ThreadPool.QueueUserWorkItem(new WaitCallback(uploadPartTask), pra);
                if (n == 0)
                    break;
                numBytesToRead -= n;
                i++;
            }
        }

        WaitHandle.WaitAll(manualEvents.ToArray());

        for (int j = 0; j < paras.Count; j++)
        {
            XElement partE = new XElement("Part");
            partE.Add(new XElement("PartNumber", paras[j].praNum));
            partE.Add(new XElement("ETag", paras[j].etag));
            Console.WriteLine("==" + paras[j].praNum + ", etag=" + paras[j].etag);
            root.Add(partE);
        }

        //所有分块上传完成后发起complete request，通知服务端合并分块
        CompleteMultipartUploadRequest completeRequest = new CompleteMultipartUploadRequest(result.getBucket(), result.getKey(), r
esult.getUploadId());
        completeRequest.setContent(new MemoryStream(System.Text.Encoding.Default.GetBytes(root.ToString())));
        CompleteMultipartUploadResult completeResult = ks3Client.completeMultipartUpload(completeRequest);
        return true;
    }

    private static void uploadPartTask(Object obj) {
        Param pra = (Param)obj;
        UploadPartRequest request = pra.request;
        PartETag tag = ks3Client.uploadPart(request);//上传本次分块内容
        if (tag.geteTag() != null) {
            pra.etag = tag.geteTag();
            pra.mrEvent.Set();
            Console.WriteLine(tag.ToString());
        }
    }

    public class Param
    {
        public ManualResetEvent mrEvent;
        public UploadPartRequest request;
        public int praNum;
        public string etag;
    }
    /**
    * 多线程分块上传例子结束
    * */
```

注：中途想停止分块上传的话请调用AbortMultipartUpload(bucketname, objectkey, uploadId);

4.14 生成带签名的URL

使用示例

```
private static void getObjUrl() {
DateTime date = DateTime.Now;
date=date.AddMinutes(5);//指定签名有效期
String url = ks3Client.generatePresignedUrl("YourBucketName", "YourKey", date);
Console.WriteLine("url:"+url);
}
```

C/C++

KS3 SDK For C/C++使用指南

注意

文档中的示例代码仅供参考之用，具体使用的时候请参考KS3 API文档，根据自己的实际情况调节参数。

目录

---

1. 概述

此SDK适用于Linux/Windows环境下C/C++版本。基于KS3 API 构建。

2. 初始化

2.1 下载源码

https://gitee.com/ks3sdk/ks3-c-sdk

2.2 获取密钥

1. 开通KS3服务，https://www.ksyun.com/user/register 注册账号 2. 进入控制台，
https://ks3.console.ksyun.com/console.html#/setting 获取AccessKeyID 、AccessKeySecret

2.3 常用术语介绍

Object（对象，文件）

在KS3中，用户操作的基本数据单元是Object。单个Object允许存储0~48.8TB的数据。 Object 包含key和data。其中，key是Object的名字；data是Object 的数据。key为UTF-8编码，且编码后的长度不得超过1024个字符。

Key（文件名）

即Object的名字，key为UTF-8编码，且编码后的长度不得超过1024个字符。Key中可以带有斜杠，当Key中带有斜杠的时候，将会自动在控制台里组织成目录结构。

其他术语请参考概念与术语

3. 快速入门

- 请先阅读 常用概念术语文档
- 常见示例请参考源码中example.c文件

3.1 创建一个bucket

```
const char* host = "ks3-cn-beijing.ksyuncs.com";
const char* bucket = "YOUR_BUCKET";
    const char* ak = "YOUR_ACCESS_KEY";
    const char* sk = "YOUR_SECRET_KEY";
```

```
int error;
buffer* resp = NULL;

resp = create_bucket(host, bucket, ak, sk, NULL, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.2 删除一个bucket

```
const char* host = "ks3-cn-beijing.ksyuncs.com";
const char* bucket = "YOUR_BUCKET";
    const char* ak = "YOUR_ACCESS_KEY";
    const char* sk = "YOUR_SECRET_KEY";

int error;
buffer* resp = NULL;

resp = delete_bucket(host, bucket, ak, sk, NULL, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.3 列出用户所有空间

```
const char* host = "ks3-cn-beijing.ksyuncs.com";
const char* ak = "YOUR_ACCESS_KEY";
const char* sk = "YOUR_SECRET_KEY";

int error;
buffer* resp = NULL;

resp = list_all_bucket(host, ak, sk, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.4 上传文件

```
const char* host = "ks3-cn-beijing.ksyuncs.com";
const char* bucket = "YOUR_BUCKET";
const char* object_key = "YOUR_OBJECT_KEY";
const char* filename = "LOCAL_DISK_FILE_PATH";
const char* ak = "YOUR_ACCESS_KEY";
const char* sk = "YOUR_SECRET_KEY";
const char* headers = "x-kss-acl:public-read";

int error;
buffer* resp = NULL;

resp = upload_file_object(host, bucket,
    object_key, filename, ak, sk, NULL, headers, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
```

```
}
buffer_free(resp);
```

3.5 分块上传

3.5.1 初始化分块上传

```
const char* host = "ks3-cn-beijing.ksyuncs.com";
const char* bucket = "YOUR_BUCKET";
const char* object_key = "YOUR_OBJECT_KEY";
const char* ak = "YOUR_ACCESS_KEY";
const char* sk = "YOUR_SECRET_KEY";
const char* headers = "x-kss-acl:public-read";

int error;
buffer* resp = NULL;

resp = init_multipart_upload(host, bucket, object_key, ak, sk, NULL, headers, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code = %ld\n", resp->status_code);
    printf("status msg = %s\n", resp->status_msg);
    if (resp->body != NULL) {
       printf("error msg = %s\n", resp->body);
    }
}
buffer_free(resp);

// 从response body中解析出uploadid:
char uploadid_str[128];
char *oid_beg_ptr = strstr(resp->body, "<UploadId>");
if (oid_beg_ptr) {
   oid_beg_ptr += strlen("<UploadId>");
   char *oid_end_ptr = strstr(oid_beg_ptr, "</UploadId>");
   if (oid_end_ptr) {
      strncpy(uploadid_str, oid_beg_ptr, oid_end_ptr - oid_beg_ptr);
      uploadid_str[oid_end_ptr - oid_beg_ptr] = 0;
  }
}
```

3.5.2 上传分块数据

```
char* data_buf = "hello world";
int data_len = strlen(data_buf);
int partNumber = 1;
resp = upload_part(host, bucket, object_key, ak, sk,
uploadid_str, partNumber, data_buf, data_len, NULL, NULL, &error);
if (error != 0) {
   printf("curl err=%d\n", error);
} else {
   printf("status code = %ld\n", resp->status_code);
   printf("status msg = %s\n", resp->status_msg);
   if (resp->body != NULL) {
      printf("error msg = %s\n", resp->body);
   }
}
buffer_free(resp);
```

3.5.3 完成分块上传

```
// 构造request body （如下是一块的例子，如果多块，则Part部分重复拼接多次）
snprintf(com_xml, sizeof(com_xml), "<CompleteMultipartUpload>\n"
           "<Part>\n"
           "<PartNumber>%d</PartNumber>\n"
           "<ETag>\"%.*s\"</ETag>"
           "</Part>\n</CompleteMultipartUpload>",
           1, 32, etag_ptr);

resp = complete_multipart_upload(host, bucket, object_key, ak, sk,
uploadid_str, com_xml, strlen(com_xml), NULL, NULL, &error);
if (error != 0) {
   printf("curl err=%d\n", error);
} else {
   printf("status code = %ld\n", resp->status_code);
   printf("status msg = %s\n", resp->status_msg);
   if (resp->body != NULL) {
     printf("error msg = %s\n", resp->body);
   }
}
```

```
buffer_free(resp);
```

3.5.4 列举已经上传的块

```
resp = list_multipart_uploads(host, bucket, ak, sk, uploadid_str, NULL, NULL, &error);
if (error != 0) {
   printf("curl err=%d\n", error);
} else {
   printf("status code = %ld\n", resp->status_code);
   printf("status msg = %s\n", resp->status_msg);
   if (resp->body != NULL) {
      printf("error msg = %s\n", resp->body);
   }
}
buffer_free(resp);
```

3.5.5 取消分块上传

```
resp = abort_multipart_upload(host, bucket, object_key, ak, sk, uploadId, NULL, NULL, &error);
if (error != 0) {
   printf("curl err=%d\n", error);
} else {
   printf("status code = %ld\n", resp->status_code);
   printf("status msg = %s\n", resp->status_msg);
   if (resp->body != NULL) {
      printf("error msg = %s\n", resp->body);
   }
}
buffer_free(resp);
```

3.6 带header上传文件

注意：header之间要以'\n'分隔

```
const char* headers = "x-kss-acl:public-read\nx-kss-callbackurl:http://www.callbackurl.com/";
resp = upload_file_object(host, bucket,
    object_key, filename, ak, sk, NULL, headers, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.7 下载文件

```
const char* to_save_file_name = "./local_save";
resp = download_file_object(host, bucket,
    object_key, to_save_file_name, ak, sk, NULL, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.8 删除文件

```
const char* object_key = "YOUR_OBJECT_KEY";
resp = delete_object(host, bucket, object_key, ak, sk, NULL, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.9 复制文件

```
const char* src_bucket = "SRC_BUCKET_NAME";
const char* src_object_key = "SRC_OBJECT_KEY";
const char* dst_bucket = "DST_BUCKET";
const char* dst_object_key = "DST_OBJECT_KEY";

resp = copy_object(host, src_bucket, src_object_key,
        dst_bucket, dst_object_key, ak, sk, NULL, NULL, &error);

if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
buffer_free(resp);
```

3.10 上传buf object

```
const char* buf = "hello world";
int buf_len = strlen(buf);

resp = upload_object(host, bucket,
    object_key, buf, buf_len, ak, sk, NULL, NULL, &error);
if (error != 0) {
    printf("curl err=%d\n", error);
} else {
    printf("status code=%d\n", resp->status_code);
    printf("status msg=%s\n", resp->status_msg);
    if (resp->body != NULL) {
        printf("error msg=%s\n", resp->body);
    }
}
free(buf);
buffer_free(resp);
```

安装

安装

本文介绍如何安装Android SDK及相关配置说明。

前提条件

- Android系统为2.3及以上版本
- 已开通对象存储功能

下载SDK

- 下载SDK jar包

安装SDK

SDK以jar包形式呈现。将*ks3-androidsdk-xxx.jar*放入工程libs文件下。右键选择Add as Library...，将jar包添加到项目中。

相关配置

以下介绍使用Android SDK前的相关配置。

- 申请AccessKey、SecretKey
- 权限配置

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

初始化

KS3Client 是 KS3 服务的 Android 客户端，它为调用者提供了一系列的方法进行操作、管理存储空间（Bucket）和文件（Object）等。在使用 SDK 发起对 KS3 的请求前，您需要初始化一个 KS3Client 实例，并对它进行一些必要设置。

设置Client

移动终端是一个不受信任的环境，把 AccessKeyId 和 AccessKeySecret 直接保存在终端用来加签请求，存在极高的风险。推荐使用 **自签名模式**，可根据需要选择下列两种方式中的一种：

- a. 直接利用AccessKey、SecretKey初始化（***不安全，仅建议测试时使用***）

```
// 初始化客户端认证信息，需要填写相应 ak、sk、endPoint
String ak = "***";
String sk = "***";

// 创建客户端对象
Ks3Client client = new Ks3Client(ak, sk, Activity.this);

// 设置Endpoint
ks3Client.setEndpoint("ks3-cn-beijing.ksyuncs.com");

// 配置类如果不设置，会有默认配置。
Ks3ClientConfiguration configuration = Ks3ClientConfiguration.getDefaultConfiguration();

client.setConfiguration(configuration);
```

- b. 实现授权回调（AuthListener）获取Token（***推荐使用***）

```
Ks3Client client = new Ks3Client(new AuthListener() {
    @Override
    public String onCalculateAuth(String httpMethod,
        String ContentType, String Date, String ContentMD5,
        String Resource, String Headers) {
            // 此处应由APP端向业务服务器发送post请求返回Token。
            // 需要注意该回调方法运行在非主线程
            String token = requsetToAppServer(httpMethod, ContentType,
                Date, ContentMD5, Resource, Headers);
            return token;
        }
    }, Activity.this);

// 配置类如果不设置，会有默认配置。
Ks3ClientConfiguration configuration = Ks3ClientConfiguration.getDefaultConfiguration();

client.setConfiguration(configuration);

client.setEndpoint("ks3-cn-beijing.ksyuncs.com");
```

onCalculateAuth()回调方法参数说明：

- Content-MD5 表示请求内容数据的MD5值，使用Base64编码
- Content-Type 表示请求内容的类型
- Date 表示此次操作的时间,且必须为 HTTP1.1 中支持的 GMT 格式，客户端应 **务必** 保证本地时间正确性
- CanonicalizedKssHeaders 表示HTTP请求中的以x-kss开头的Header组合
- CanonicalizedResource 表示用户访问的资源

关于授权回调的必要说明：

- 对于使用AuthListener以Token方式初始化SDK的用户，需要注意onCalculateAuth()回调方法中的参数，为计算签名的参数，服务器端应根据签名生成规则，利用AccessKeyID及AccessKeySecret **计算出签名并正确返回给SDK**。
- onCalculateAuth()回调方法的参数Content-MD5, Content-Type, CanonicalizedKssHeaders参数 **可为空**。若为空，则SDK会使用空字符串("")替代，但Date和CanonicalizedResource不能为空。
- 为保证请求时间的一致性，需要App客户端及客户业务服务器保证各自的时间正确性，否则用 **错误的时间** 尝试请求，会返回403Forbidden错误。

设置网络参数

也可以在初始化的时候设置详细的KS3ClientConfiguration：

```
Ks3ClientConfiguration configuration = Ks3ClientConfiguration.getDefaultConfiguration();

// 设置http超时时长，默认20000
```

```
configuration.setConnectionTimeout(20000);

// 设置socket超时时长，默认50000
configuration.setSocketTimeout(50000);

// 设置最大连接数，默认10
configuration.setMaxConnections(10);

// 设置代理host
configuration.setProxyHost(null);

// 设置代理用户名
configuration.setProxyUsername(null);

// 设置密码
configuration.setProxyPassword(null);

// 设置代理端口
configuration.setProxyPort(-1);

// 设置最大重连次数
configuration.setMaxRetrytime(0);

// 设置重连超时时长，默认5000
configuration.setRetryTimeOut(5000);
client.setConfiguration(configuration);
```

设置自定义user-agent

```
Ks3ClientConfiguration configuration = Ks3ClientConfiguration.getDefaultConfiguration();

// 设置自定义 user-agent, 默认使用'ks3-android-sdk'
configuration.setUserAgent("customUserAgent");
```

对 SDK 中同步接口、异步接口的一些说明

- 考虑到移动端开发场景下不允许在UI线程执行网络请求的编程规范，SDK大多数接口都提供了同步、异步两种调用方式，同步接口调用后会阻塞等待结果返回，而异步接口需要在请求时传入回调函数，请求的执行结果将在回调中处理。
- 同步接口不能在UI线程调用。遇到异常时，将直接抛出KS3ClientException或者KS3Error异常，前者指本地遇到的异常如网络异常、参数非法等；后者指KS3返回的服务异常，如鉴权失败、服务器错误等。
- 异步请求遇到异常时，异常会在回调函数中处理。
- 业务服务器计算签名方式

快速入门

快速入门

本节介绍如何快速使用云存储Android SDK完成常见操作，如创建存储空间、上传文件、下载文件等。

本工程的更多用法请参考以下两种方式：

- 查看demo目录（包含上传本地文件、下载文件、断点续传、设置回调等示例），详情请点击查看
- 直接 git clone 工程

运行本工程前，您需要配置必要参数 Config，配置 Config 示例代码如下：

```
public class Config {
    // 测试用AK&SK
    public static final String ACCESS_KEY_ID = "<yourAccessKeyId>";
    public static final String ACCESS_KEY_SECRET = "<yourAccessSecretKey>";
    // 访问的endpoint地址
    public static final String END_POINT = "<yourEndPoint>";
    // bucket名称
    public static final String BUCKET_NAME = "<bucketName>";
    // 重试时长
    public static final int  RETRY_TIMEOUT = 3000;
    // 最大重连次数
    public static final int MAX_RETRY_TIME = 10;

    // 上传到bucket的object key
    public static final String OBJECT_KEY = "test.jpg";
    // 下载到本地的文件名称
    public static final String DOWNLOAD_FILE_NAME = "formKs3.jpg";
```

```
}
```

创建存储空间

存储空间是全局命名空间，相当于数据的容器，可以存储若干文件。 以下代码用于新建一个存储空间：

```
// 创建指定名称的bucket
client.createBucket(Config.BUCKET_NAME, new CreateBucketResponceHandler() {
    @Override
    public void onFailure(int i, Ks3Error ks3Error, Header[] headers, String s, Throwable throwable) {
        System.out.println("创建失败!");
    }

    @Override
    public void onSuccess(int i, Header[] headers) {
        System.out.println("创建成功!");
    }
});
```

- 存储空间的命名规范，请参见空间管理-基本操作。
- 您可以在创建存储空间时指定存储空间的权限和存储类型。

上传文件

以下代码用于将指定的本地文件上传到OSS：

```
String filePath = Environment.getExternalStorageDirectory().getPath();
// 待上传文件
File file = new File(filePath + "/storage/watch.jpg");
// 构造上传请求
client.putObject(Config.BUCKET_NAME, Config.OBJECT_KEY , file, new PutObjectResponseHandler() {
    @Override
    public void onTaskFailure(int i, Ks3Error ks3Error, Header[] headers, String s, Throwable throwable) {
        // 上传失败 请求异常
        System.out.println("上传失败!");
    }

    @Override
    public void onTaskSuccess(int i, Header[] headers) {
        System.out.println("上传成功!");
    }

    @Override
    public void onTaskStart() {
    // 异步任务开启
    }

    @Override
    public void onTaskFinish() {
    // 任务结束
    }

    @Override
    public void onTaskCancel() {
    // 任务取消 可通过client.cancel(MainActivity.this);进行取消上传
    }

    @Override
    public void onTaskProgress(double v) {
    // 任务进度 这里可以得到上传的进度
    }
});
```

下载指定文件

以下代码用于将指定的OSS文件下载到本地文件：

```
// 下载文件的存放位置
String filePath = Environment.getExternalStorageDirectory().getPath();
File destFile = new File(filePath + "/storage/" + Config.DOWNLOAD_FILE_NAME);
// 请求下载指定的文件
client.getObject(this, Config.BUCKET_NAME, Config.OBJECT_KEY, new GetObjectResponseHandler(destFile, false) {
    @Override
    public void onTaskProgress(double v) {
    // 下载的进度
    }

    @Override
    public void onTaskStart() {
```

```
    // 下载任务开始
    }

    @Override
    public void onTaskFinish() {
    // 下载任务完成
    }

    @Override
    public void onTaskCancel() {
    // 下载任务取消 取消操作与取消上传是一样的
    }

    @Override
    public void onTaskSuccess(int i, Header[] headers, GetObjectResult getObjectResult) {
        System.out.println("下载成功");
    }

    @Override
    public void onTaskFailure(int i, Ks3Error ks3Error, Header[] headers, Throwable throwable, File file) {
        System.out.println("下载失败");
    }
});
```

存储空间

上传文件

下载文件

管理文件

SDK异常处理

注意：下面的说明描述以JAVA SDK的使用为例 ,其它SDK可参考此文档。

常见异常说明

错误码	描述	状态码	如何处理
BucketAlreadyExists	桶已存在	409	桶名有冲突，需要重新命名
BucketNotEmpty	桶不为空	409	先删除桶内文件，再执行删除
ClientIllegalArgument	桶名不合法		
NoSuchBucket	Bucket不存在	404	Bucket不存在，检查是否拼写错误、未创建Bucket
NotFound	文件不存在	404	检查文件名是否拼写错误、未上传、已删除
AccessDenied	没有权限，拒绝访问	403	1、检查是否使用匿名的方式访问私密文件；2、如果用子账号或者其他账号访问资源，检查是否给子账号或者其他账号进行了正确的授权；3、检查是否使用自定义域名访问，但是在KS3未绑定域名
RequestTimeTooSkewed	发起请求的时间和服务器时间超出15分钟	403	检查机器时钟
InvalidAccessKeyId	AK非法	403	检查AK是否拼写错误
SignatureDoesNotMatch	签名不匹配	403	1、检查SK是否正确；2、检查签名算法
InvalidKey	文件名不合法	400	检查文件名是否不合法
InvalidDigest	md5错误	400	检查上传的md5值是否错误

InvalidPartNum	上传分块，块号错误	400	块号范围[1,10000]
InvalidPartOrder	分块上传完成，块号顺序错误	400	块号不连续，有缺失块
EntityTooLarge	单次上传太大，超过5G	400	单次上传太大，超过5G
UnknownHost	域名未解析		1、检查域名是否拼写有误；2、检查公网网络是否异常
CallRemoteFail	网络异常		使用重试机制，见下面的说明

网络异常时的重试机制

注意：对网络异常可以进行有限次数的重试，对业务异常（KS3返回的）不建议进行重试，需要客户按照返回的异常错误码，查找原因。

以JAVA SDK的putObject为例：

```
String bucketName = "test-bucket";
String objectKey = "test-key";
File file = new File("d:\\test");
try{
    client.putObject(bucketName, objectKey, file);
}catch(CallRemoteFailException e){
    if(e.getMessage().indexOf("ConnectTimeoutException")>-1||e.getMessage().indexOf("SocketTimeoutException")>-1){
        System.out.println("newwork error,need retry:"+bucketName+" objectKey:"+objectKey);
        //进行重试，对于网络异常，建议进行有限次数的重试
        //client.putObject(bucketName, objectKey, file);
}
}catch(Ks3ServiceException e){
    System.out.println(e.getStatueCode());//http status
    System.out.println(e.getErrorCode());//error code
    System.out.println(e.getErrorMessage());//error message
    System.out.println(e.getRequestId());//requestid，请求唯一标识，可以将此标识提供后售后人员，协助排查
}finally{
    //release resource
}
```