盛大云存储 Java 语言 SDK 使用说明

目录

1. 系统要求

使用盛大云存储的 java sdk 需要 jdk1.5 或以上支持，在使用前请检查您的系统。以下的类库 commons-lang、commons-codec、commons-pool、com.google.guava、org.bouncycastle、log4j、org.slf4j、joda-time、org.apache.httpcomponents、junit、org.mockito 在使用 sdk 的时候会被用到，请注意配置您的环境。我们推荐使用 maven 进行包管理，sdk 内部已经包含了 pom.xml

2. 盛大云存储的基本概念

2.1 AccessKey

AccessKey 由盛大云存储单独颁发。 AccessKey 在所有的操作中都会被使用，并且会以明文形式传输。用于标识用户身份。每位用户一个，不会重复。

AccessKey 通过云计算网站的云存储用户信息管理获得：http://www.grandcloud.cn（需要登录）。

2.2 SecretAccessKey

SecretAccessKey 也由盛大云存储颁发，SecretAccessKey 总是随同 AccessKey 一起分发，一个 AccessKey 对应一个 SecretAccessKey。

SecretAccessKey 通过云计算网站的云存储用户信息管理获得：http://www.grandcloud.cn（需要登录）。

出于安全问题的考虑，对云存储的所有的操作都需要由 SecretAccessKey 签名摘要后才能有效，摘要信息将成为请求的一部分，发送给云系统。

任何时候 SecretAccessKey 都不应发送给盛大云存储系统。

SecretAccessKey 涉及您存储资料的安全问题，所以请妥善保存您的 SecretAccessKey，不要泄漏给第三方。如 SecretAccessKey 发生泄漏，请立即登录盛大云计算网站，云存储用户信息管理，将原 SecretAccessKey 作废，重新生成。

2.3 Bucket

在用户空间内，用户根据需要可以建立不同的 Bucket。

你可以把 Bucket 想象成文件系统内的目录，在盛大云存储系统中一个用户空间内最多只能有 **100** 个 Bucket。

Bucket 命名全局唯一，也就是说所有盛大云存储的用户的 Bucket 都是不一样的。 例如有 A 用户建立了名为”aaa”的 Bucket，此时 B 用户希望创建名字同样为”aaa”的 Bucket 将会失败。

2.4 Bucket 的命名规则

a) 由小写字母或数字或点号(.) 或下划线(_)或破折号(-)组合而成。

b) 开头必须是 数字或者小写字母。
c) 长度必须 大于等于 3 字节 小于等于 255 字节
d) 不能是一个 IP 地址形式。比如 192.168.1.1 这样的格式是不允许的
e) 不能以 snda 作为 Bucket 的开头
f) 如果希望以后提供 DNS 解析，则 Bucket 命名还需符合 DNS 主机名的命名规则

2.5 Object

Object 是盛大云存储的主要对象。用户存储的内容都以 Object 形式存储在系统里。
1 个 Object 必须存储在盛大云存储系统的某个 Bucket 内。
1 个 Object 包含了 **ObjectName**，**ObjectMetadata** 以及 **ObjectData** 3 个部分。

ObjectName 就是 Object 的名字，在同一个 Bucket 下的 ObjectName 是唯一的。

2.6 ObjectName 的命名规则

a) 使用 Utf-8 编码规则
b) ObejctName 的长度大于等于 1 字节小于等于 1024 字节

3. 基本数据类型

3.1 BaseStorageItem

云存储数据对象的抽象类型，StorageBucket 和 StorageObject 由其派生而来，该类型含有操作数据对象的 Metadata 一系列方法，主要包含：add, get, contains, remove, clean, replace 等。

3.2 StorageBucket

Bucket 对象的基础类型，是 BaseStorageItem 的子类，可以通过 bucket name 指定构造该名称的 bucket 对象，该类型包含手动设置 Bucket 创建时间的方法。

3.3 StorageObject

Object 对象的基础类型，同样是 BaseStorageItem 的子类，在创建该类型对象时，objectKey 代表 object 在存储中的名字，可以指定实际指向的数据对象，根据构造函数的不同，可以接受 File，InputStream，Byte[]三种类型的数据参数，也可以在创建时不指定实际的数据，再通过 set 方法将对象指向实际的数据；在对象使用结束后，需要通过 closeDataInputStream 方法关闭使用过的数据对象。

StorageObject 类型提供的针对数据的方法有：获得数据长度(Content length)，获取数据 MD5(hex 或 hex as base64)，指定或获取数据类型(Content type)，指定最后修改时间(Last modified date)，验证数据的正确性(data verification)。

3.4 CSBucket

盛大云存储的 Bucket 对象

3.5 CSObject

盛大云存储的 Object 对象

4. 存储服务对象

4.1 StorageService

云存储服务基本类型，抽象类，主要负责控制与存储服务的交互，提供了当前存储服务的全部服务 api。根据服务 api，派生出基于 REST 服务的类型：RestStorageService。创建存储服务对象时，都需要指定身份验证对象传入构造函数中，可以通过 isAuthenticatedConnection 方法验证是否已经存在身份验证信息。该存储类型提供的存储服务的 api 有：列出用户的所有 bucket(List all my bucket)， 查看某个 bucket 信息(get bucket)，在存储服务中创建 bucket(create bucket)，查看某个 object 是否在该 bucket 中(is object in bucket)，上传 object(put object)，下载 object(get object)，查看 object 的 metadata(head object)，删除 object(delete object)，关闭当前服务对象(shut down)。

4.2 RestStorageService

云存储服务 api 的 rest 实现

4.3 CSService and RestCSService

盛大云存储服务类型，分别由 StorageService 和 RestStorageService 派生来的子类。主要用于描述盛大云存储服务的类型。

5. 身份验证对象

5.1 SNDACredentials

SNDA 身份验证对象，构造时需要有云存储服务提供给用户的 access key 和 secret key。

5.2 保存 Credential

EncryptionUtil 类提供了保存并加密 Credential 的 API，可以通过执行 SNDACredentials 的 main 函数来启动 Credential 的加密进程。

加密程序启动前中需指定以下启动参数：<User Name>, <File Path>

```
User Name: A human-friendly name for the owner of the credentials, e.g.
```

Horace."
File Path: Path and name for the encrypted file. Will be replaced if it already exists.
加密算法默认以PBEWithMD5AndDES方式加密

5.3 读取 Credential

通过 ProviderCredential 的 load 方法，可以从已保存的 Credential 文件中读取数据，并返回一个 ProviderCredential 对象，可以用做实例化存储服务对象的构造函数参数。

6. 使用说明

6.1 创建云存储服务对象

```
ProviderCredentials credentials =
                new SNDACredentials("xxxxx", "xxxxxxxxxxxxxx");
CSService service = new RestCSService(credentials);
```

6.2 获得用户 Bucket 列表

```
CSBucket[] buckets = service.listAllBuckets();
```

6.3 创建 Bucket

```
CSBucket csBucket = new CSBucket("testBucket");
service.createBucket(bucket);
```

6.4 获取 Bucket 信息

```
CSBucket csBucket = service.getBucket("testBucket");
```

6.5 删除 Bucket

```
service.deleteBucket("testBucket");
```

6.6 上传 Object

```
CSObject csObject = new CSObject(new File("folder/file"));
service.putObject("testBucket", object);
```

6.7 获取 Object 的 Metadata

```
CSObject csObject =
                service.headObject("testBucket", "testObject");
```

6.8 下载 Object

```
CSObject csObject = service.getObject("testBucket", "testObject");
```

6.9 删除 Object

```
service.deleteObject("testBucket", "testObject");
```

6.10 拷贝 Object

```
StorageObject sourceObject = new CSObject(new
File("resources/mime.types"));
service.putObject("testBucket", object);
StorageObject destObject = new StorageObject("mime.types.copy");
service.copyObject("testBucket", sourceObject.getKey(),
"testBucket2", destObject, true);
```

6.11 移动 Object

```
service.moveObject("testBucket", "testObject",
              "testBucket2", destObject, true);
```

6.12 重命名 Object

```
service.renameObject("testBucket", "testObject", destObject);
```

6.13 判断 Object 是否在 Bucket 中

```
boolean objectInBucket =
              service.isObjectInBucket(
                     "testBucket", "testObject");
```

6.14 获取 Bucket，如果不存在则创建

```
CSBucket csBucket = service.getOrCreateBucket("testBucket");
```

6.15 获取 Bucket location

```
String location = service.getBucketLocation("testBucket");
```

6.16 获取 Bucket 下 Object 列表

```
CSObject[] objects = service.listObjects("testBucket");
```

6.17 设置 HTTPS 访问

```
service. setHttpsOnly(true);
```

默认状态是通过HTTP方式访问的

6.18 获取 Bucket policy

```
policyDocument是描述bucket policy的xml字符串
service.setBucketPolicy("testBucket", policyDocument);
```

6.19 设置 Bucket policy

```
String policy = service.getBucketPolicy("testBucket");
```

6.20 删除 Bucket policy

```
service.deleteBucketPolicy("testBucket");
```

6.21 指定开始一个 MultiUpload

```
MultipartUpload multipart =
              service.multipartStartUpload(
                     "testBucket" ,"multiupload_key", metadata);
```

6.22 上传一个 MultiUpload 对象的 part

```
multipart表示一个MultiUpload对象的part, partNumber表示这个part的序号是第几部分(从1开始计数), partObject是需要上传的CSObject对象
service.multipartUploadPart(multipart, partNumber, partObject);
```

6.23 列出 Bucket 下所有 MultiUpload 对象的 part

```
multipart表示一个MultiUpload对象的part
service.multipartListParts(multipart);
```

6.24 完成 MultiUpload 操作

```
multipart表示一个MultiUpload对象的part
service.multipartCompleteUpload(multipart);
```

6.25 取消 MultiUpload 操作

```
multipart表示一个MultiUpload对象的part
service.multipartAbortUpload(multipart);
```

6.26 创建带签名的 URL

```
String signedPutUrl= service.createSignedPutUrl("testBucket",
              "testObject", headersMap, expireDate);

String signedGetUrl = service.createSignedGetUrl("testBucket",
              "testObject", expireDate);

String signedHeadUrl = service.createSignedHeadUrl("testBucket",
              "testObject", expireDate);

String signedDeleteUrl =
          service.createSignedDeleteUrl("testBucket",
              "testObject", expireDate);
```

6.27 使用带签名的 URL 上传 Object

```
CSObject csObject = new CSObject("testObject", "data content");
csObject.addMetadata("Date", expireDate);
String signedPutUrl = service.createSignedPutUrl(
          "testBucket", "testObject",
          csObject.getMetadataMap(), expireDate);
service.putObjectWithSignedUrl(signedPutUrl, csObject);
```

6.28 使用带签名的 URL 下载 Object

```
String signedGetUrl = service.createSignedGetUrl("testBucket",
              "testObject", expireDate);
CSObject csObject = service.getObjectWithSignedUrl(signedGetUrl);
```

6.29 使用带签名的 URL 获取 Object 的 Metadata

```
String signedHeadUrl = service.createSignedHeadUrl("testBucket",
              "testObject", expireDate);
CSObject csObject =
          service.getObjectDetailsWithSignedUrl(signedHeadUrl);
```

6.30 使用带签名的 URL 删除 Object

```
String signedDeleteUrl =
              service.createSignedDeleteUrl(
                     "testBucket",
                     "testObject", expireDate);
service.deleteObjectWithSignedUrl(signedDeleteUrl);
```

7. 异常

7.1 ServiceException

在访问云存储过程中，所有没有能够正常完成服务请求的操作，都会返回 ServiceException，该 Exception 是由 RuntimeException 派生而来，ServiceException 的对象中，保存了以下 HTTP 请求失败返回的全部信息：response code，response status，response date，response body，response header，以及这次 request 的信息：request method， request path， request host；并对错误返回的 response body 进行了解析，并会得出以下由存储服务器获得到的错误返回的响应：error code， error message， error request id， error resource。例如在数据 MD5 不正确时，我们执行的代码如下：

```
    // Create test object with an MD5 hash of the data.
    String dataString = "Text for MD5 hashing...";
    StorageObject object = new StorageObject("Testing MD5 Hashing",
dataString);
    object.setContentType("text/plain");

    // Calculate hash data for object.
    byte[] md5Hash =
ServiceUtils.computeMD5Hash(dataString.getBytes());

    // Ensure that using an invalid hash value fails.
    try {
        object.addMetadata("Content-MD5", "123");
        service.putObject("testBucket", object);
    } catch (ServiceException e) {
        System.out.println(e.getErrorCode()
                + "\n" + e.getErrorMessage()
                + "\n" + e.getErrorResource()
                + "\n" + e.getErrorRequestId());
    }
```

返回如下：

BadDigest

The Content-MD5 you specified did not match what we received.

/testBucket/Testing MD5 Hashing

e275a310-0af7-4cb0-a491-aef3aca53c75

8. JavaDoc

更详细的 API 说明请参见 JavaDoc.

如果在使用中遇到任何问题，请在 **http://forum.grandcloud.cn/** 反馈，我们将及时跟进。谢谢！