# 文件同步器设计说明书

班级：18级软件工程1班 姓名：许航

**一、项目介绍**

基于通过S3 Browser和编程的方式实现S3对象存储的使用、版本控制、分片上传&下载、与Lambda结合使用、访问权限设置、文件属性更改、定期删除等基础操作，个人设计一个简单的同步器综合应用所学编程方式实现本地文件夹与S3 Browser文件同步。文件同步器的基本要求如下：

1、指定本地某个目录与S3的某个bucket进行文件同步。

2、程序启动时把bucket中的文件同步到本地。需要处理文件冲突。

3、本地添加、修改了文件，需要上传到S3；本地删除了文件，也要删除S3上对应的文件。

4、对于超过20MB的文件，需要使用分块上传/下载，传输中断、程序重启可以继续原来的进度。

**二、总体设计**

1、文件同步器总体设计

文件同步器可以分成 从S3 bucket同步到本地文件夹，从本地文件夹同步到S3 bucket两个模块。

a. 从S3 bucket同步到本地文件夹

程序启动时，可以选择将S3 bucket文件同步到本地文件夹中，如存在相同的文件则不予下载至本地文件夹；

如本地文件夹存在S3 bucket中不存在的文件，则予以删除操作；

对于超过20MB的文件，则使用分块下载的方式下载文件至本地文件夹。

b. 从本地文件夹同步到S3 bucket

程序启动时，可以选择将本地文件夹同步到S3 bucket中，如存在相同的文件则进行判断文件是否有更新，如有更新则上传该文件的新版本至S3 bucket，如无更新则不予上传。

如本地文件夹中存在S3 bucket不存在的文件，则予以上传至S3 bucket。

如S3 bucket中存在本地文件夹已经删除的文件，则在至S3 bucket删除该文件。

2、系统环境配置

软件：JDK 1.8、Eclipse IDE for Java、aws-java-sdk.zip、S3 Browser

操作系统：windows 10

**三、详细设计**

本程序仅由bingocloud-sdk-s3-homework java项目中Main.java以及其依赖包驱动。（由于依赖包过大不予上传至github代理）主要包括bucket2Local()和local2Bucket()两个功能函数。

1.Main.java

主要包括S3Client的创建以及其授权信息的声明，对两个功能函数的综合控制，合理化同步器选择功能。

2.bucket2Local()

a.删除本地目录中不存在于bucket的文件

通过对比本地目录中所有文件路径以及bucket中所有文件下载路径与keyName合成的路径，判断前者（本地）中是否存在后者（bucket）不包括的路径，将其从本地文件夹中删除。

调用[java](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava).[io](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava.io).[File](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava.io(File.class%E2%98%83File).delete()删除本地文件。

b.下载本地目录中不存在的bucket文件

同a中的对比，判断后者（bucket）中是否存在前者（本地）不包括的路径，将其下载至本地文件夹。

调用download()函数直接下载小于20M的文件。

1. **private** **static** **void** download(AmazonS3 s3,String keyName,String path) {
2. System.out.format("Downloading %s to S3 bucket %s...\n", keyName, bucketName);
4. S3ObjectInputStream s3is = **null**;
5. FileOutputStream fos = **null**;
6. **try** {
7. S3Object o = s3.getObject(bucketName, keyName);
8. s3is = o.getObjectContent();
9. fos = **new** FileOutputStream(**new** File(filePath));
10. **byte**[] read\_buf = **new** **byte**[64 \* 1024];
11. **int** read\_len = 0;
12. **while** ((read\_len = s3is.read(read\_buf)) > 0) {
13. fos.write(read\_buf, 0, read\_len);
14. }
15. } **catch** (AmazonServiceException e) {
16. System.err.println(e.toString());
17. System.exit(1);
18. } **catch** (IOException e) {
19. System.err.println(e.getMessage());
20. System.exit(1);
21. } **finally** {
22. **if** (s3is != **null**) **try** { s3is.close(); } **catch** (IOException e) { }
23. **if** (fos != **null**) **try** { fos.close(); } **catch** (IOException e) { }
24. }
26. System.out.println("Done!");

29. }

c.对于本地目录已存在bucket中的文件，则不下载

通过同a中的对比后，选择出路径一致的文件，再对比相同路径文件的Etag判断该文件是否是相同的文件，相同则不予下载操作，不相同则下载该文件至本地文件夹。

调用calculateETag()计算本地文件的Etag值，bucket中文件则通过调用S3ObjectSummary类的方法getETag()或者类S3Obejec的方法 s3Object.getObjectMetadata().getETag()即可获取。

1. **private** **static** String calculateETag(String localFilePath) {
2. File file = **new** File(localFilePath);
3. String s = "";
4. **try** {
5. InputStream is = **new** FileInputStream(file);
6. String digest = DigestUtils.md5Hex(is); // 计算MD5值
7. s = digest;
8. } **catch** (FileNotFoundException e) {
9. // TODO Auto-generated catch block
10. e.printStackTrace();
11. } **catch** (IOException e) {
12. // TODO Auto-generated catch block
13. e.printStackTrace();
14. }
15. **return** s;
16. }

同b中如文件大小小于20M，则调用download()函数直接下载Etag不一致的文件至本地文件夹。

\*.对于大于20M的文件使用分片下载的方式下载至本地文件夹。

调用downloadBigFile()函数。

1. **private** **static** **void** downloadBigFile(AmazonS3 s3,String keyName,String savePath) {
2. System.out.format("Downloading %s to S3 bucket %s...\n", keyName, bucketName);
3. File file = **new** File(savePath);
5. S3Object o = **null**;
7. S3ObjectInputStream s3is = **null**;
8. FileOutputStream fos = **null**;
10. **try** {
11. // Step 1: Initialize.
12. ObjectMetadata oMetaData = s3.getObjectMetadata(bucketName, keyName);
13. **final** **long** contentLength = oMetaData.getContentLength();
14. **final** GetObjectRequest downloadRequest = **new** GetObjectRequest(bucketName, keyName);
16. fos = **new** FileOutputStream(file);
18. // Step 2: Download parts.
19. **long** filePosition = 0;
20. **for** (**int** i = 1; filePosition < contentLength; i++) {
21. // Last part can be less than 5 MB. Adjust part size.
22. partSize = Math.min(partSize, contentLength - filePosition);
24. // Create request to download a part.
25. downloadRequest.setRange(filePosition, filePosition + partSize);
26. o = s3.getObject(downloadRequest);
28. // download part and save to local file.
29. System.out.format("Downloading part %d\n", i);
31. filePosition += partSize+1;
32. s3is = o.getObjectContent();
33. **byte**[] read\_buf = **new** **byte**[64 \* 1024];
34. **int** read\_len = 0;
35. **while** ((read\_len = s3is.read(read\_buf)) > 0) {
36. fos.write(read\_buf, 0, read\_len);
37. }
38. }
40. // Step 3: Complete.
41. System.out.println("Completing download");
43. System.out.format("save %s to %s\n", keyName, filePath);
44. } **catch** (Exception e) {
45. System.err.println(e.toString());
47. System.exit(1);
48. } **finally** {
49. **if** (s3is != **null**) **try** { s3is.close(); } **catch** (IOException e) { }
50. **if** (fos != **null**) **try** { fos.close(); } **catch** (IOException e) { }
51. }
52. System.out.println("Done!");
53. }

3.local2Bucket()

通过2中的方法同样对比地目录中所有文件路径以及bucket中所有文件下载路径与keyName合成的路径。

将所有文件路径分成3种：本地目录新添加的文件路径，本地目录和bucket均存在但本地文件存在更新的文件路径，本地目录新删除的文件路径，对应a.b.c.中的3中操作。

a.上传本地目录中存在但bucket中不存在的文件

判断本地目录所有文件路径是否包含在bucket所有合成路径中，选择不被包含的本地目录文件路径。

通过[java](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava).[io](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava.io).[File](eclipse-javadoc:%E2%98%82=bingocloud-sdk-s3-homework/D:%5C/Program%20Files%5C/Java%5C/jdk-14.0.1%5C/lib%5C/jrt-fs.jar%60java.base=/module=/true=/%3Cjava.io(File.class%E2%98%83File).length()判断文件大小是否小于20M，若大于则调用uploadBigFile()函数。

调用upload()函数。

1. **private** **static** **void** upload(AmazonS3 s3,String path) {
2. **final** String keyName = path.substring(filePath.length());
3. **final** File file = **new** File(path);
5. **if**(file.length()>20<<20) {
6. uploadBigFile(s3,path);
7. }
8. **for** (**int** i = 0; i < 2; i++) {
9. **try** {
10. s3.putObject(bucketName, keyName, file);
11. **break**;
12. } **catch** (AmazonServiceException e) {
13. **if** (e.getErrorCode().equalsIgnoreCase("NoSuchBucket")) {
14. s3.createBucket(bucketName);
15. **continue**;
16. }
18. System.err.println(e.toString());
19. System.exit(1);
20. } **catch** (AmazonClientException e) {
21. **try** {
22. // detect bucket whether exists
23. s3.getBucketAcl(bucketName);
24. } **catch** (AmazonServiceException ase) {
25. **if** (ase.getErrorCode().equalsIgnoreCase("NoSuchBucket")) {
26. s3.createBucket(bucketName);
27. **continue**;
28. }
29. } **catch** (Exception ignore) {
30. }
32. System.err.println(e.toString());
33. System.exit(1);
34. }
35. }
37. System.out.println("Done!");
39. }

b.对于本地目录和bucket中均存在的文件，如本地目录文件更新则上传至bucket中，如本地目录文件未更新，则不上传

选择本地目录文件路径与bucket合成路径一致的文件，进行Etag值的对比以判断文件是否进行了更新。

同1.b 调用calculateETag()计算本地文件的Etag值，bucket中文件则通过调用S3ObjectSummary类的方法getETag()或者类S3Obejec的方法 s3Object.getObjectMetadata().getETag()即可获取。

如果Etag值不一致，则将该文件上传新版本，同样调用2.a中的upload()函数。

c.删除本地目录不存在的但bucket中存在的文件

判断bucket所有合成路径是否包含在本地目录的所有文件路径中，选择不被包含的bucket合成路径。

调用delete()函数删除bucket中多余的文件。

1. **private** **static** **void** delete(AmazonS3 s3,List<String> Deletepaths) {
2. **for**(String path:Deletepaths) {
3. System.out.format("Deleting S3 bucket %s file %s...\n", bucketName,path);
4. **final** String keyName = path.substring(filePath.length());
5. s3.deleteObject(bucketName,keyName);
6. System.out.println("Done!");
7. }
8. }

\*对于超过20M的文件采用分片上传的方式上传至bucket。

调用uploadBigFile()函数。

1. **private** **static** **void** uploadBigFile(AmazonS3 s3,String path) {
3. // Create a list of UploadPartResponse objects. You get one of these
4. // for each part upload.
5. ArrayList<PartETag> partETags = **new** ArrayList<PartETag>();
6. File file = **new** File(path);
7. **final** String keyName = path.substring(filePath.length());
8. **long** contentLength = file.length();
9. String uploadId = **null**;
11. **try** {
12. // Step 1: Initialize.
13. InitiateMultipartUploadRequest initRequest =
14. **new** InitiateMultipartUploadRequest(bucketName, keyName);
15. uploadId = s3.initiateMultipartUpload(initRequest).getUploadId();
16. System.out.format("Created upload ID was %s\n", uploadId);
18. // Step 2: Upload parts.
19. **long** filePosition = 0;
20. **for** (**int** i = 1; filePosition < contentLength; i++) {
21. // Last part can be less than 5 MB. Adjust part size.
22. partSize = Math.min(partSize, contentLength - filePosition);
24. // Create request to upload a part.
25. UploadPartRequest uploadRequest = **new** UploadPartRequest()
26. .withBucketName(bucketName)
27. .withKey(keyName)
28. .withUploadId(uploadId)
29. .withPartNumber(i)
30. .withFileOffset(filePosition)
31. .withFile(file)
32. .withPartSize(partSize);
34. // Upload part and add response to our list.
35. System.out.format("Uploading part %d\n", i);
36. partETags.add(s3.uploadPart(uploadRequest).getPartETag());
38. filePosition += partSize;
39. }
41. // Step 3: Complete.
42. System.out.println("Completing upload");
43. CompleteMultipartUploadRequest compRequest =
44. **new** CompleteMultipartUploadRequest(bucketName, keyName, uploadId, partETags);
46. s3.completeMultipartUpload(compRequest);
47. } **catch** (Exception e) {
48. System.err.println(e.toString());
49. **if** (uploadId != **null** && !uploadId.isEmpty()) {
50. // Cancel when error occurred
51. System.out.println("Aborting upload");
52. s3.abortMultipartUpload(**new** AbortMultipartUploadRequest(bucketName, keyName, uploadId));
53. }
54. System.exit(1);
55. }
56. System.out.println("Done!");
58. }

**四、总结**

通过编写简易文件同步器，对本周实训所学习的S3 brower 中S3对象存储的使用、版本控制、分片上传&下载等基础操作有更深刻的理解，了解到文件的唯一标识Etag的作用，对代码编写有较大的帮助。

仔细阅读实训操作手册并结合网上查找的Amazon S3 官方文档更好地应用依赖库中所提供的类和函数，对大数据的存储有了一定的了解。实训操作手册很详细分类也比较清晰，比网上查找的资料易于应用，为本次大作业提供了很大的帮助。