设计说明书

引言

本程序的目的是编写一个本地文件夹与S3云端桶的简单文件同步器。

文档简介

读者对象

- 用户：老师。
- 开发者：学生（我）。

软件总体说明

该软件属于作业任务驱动型的功能软件，基本基于命令行来完成相关任务，所涉及的设计功能有：

- 指定本地某个目录与S3的某个Bucket进行文件同步；
- 程序启动时把Bucket中的文件同步到本地，并需要处理文件冲突；
- 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；
- 对于超过20MB的文件，需要使用分块上传/下载，传输中断、程序重启也可以继续原来的进度。

对一些名词的解释：

- 文件冲突：本地存在与云端同名的文件，即视为产生文件冲突。
- 文件修改：监测文件的最后修改时间，若本次同步与上次同步时间不同，即视为文件被修改。
- 分块上传下载：分出的每个块为5MB。

模块设计与功能说明

根据软件设计的任务总体说明，可以将软件的设计分为三个部分：

1. 指定本地目录、指定S3的云端桶；
2. 程序启动时即进行把Bucket中的文件同步到本地指定文件夹，并处理文件冲突；
3. 本地对同步文件夹中的文件的添加、修改、删除都同步到云端。

指定本地目录和S3云端桶

在本程序中直接通过控制台输入获取。

```
        //  控制台获取要同步的本地目录和s3上的桶
        System.out.print("请输入要同步的本地目录：");
//        String synLocalPath = scanner.nextLine();
        String synLocalPath = "G:/Bingo Big Data Development/synchronization
directory/";
        System.out.println("\n同步的本地目录：" + synLocalPath);

        System.out.print("请输入要同步的s3存储桶：");
//        String bucketName = scanner.nextLine();
        String bucketName = "luna";
        System.out.println("\n同步的s3存储桶：" + bucketName);
```

程序启动时的云端Bucket的文件在本地同步

这里包括了对大文件的分块上传、下载，以及对于文件冲突的处理方式，具体可见注释。

首先，在程序启动的时候，会扫描S3云端Bucket中的所有文件，然后判断是否是文件夹。如果是文件夹，且本地不存在该文件夹的话，就在本地新建该文件夹；不是文件夹，是文件的话，将会判断本地是否有重名文件，没有就直接下载，有的话，就会给出两个选项，分别是保留本地文件覆盖云端文件，或者是下载云端文件覆盖本地文件。

在下载云端文件覆盖本地文件部分，实现了断点续传的下载。

```
        //获取s3中的对象，路径从根目录开始
        List<String> objectPathList = new ArrayList<String>();
        ObjectListing objects = s3.listObjects(bucketName);
        do {
            for (S3ObjectSummary objectSummary : objects.getObjectSummaries()) {
                System.out.println("Object: " + objectSummary.getKey());
                objectPathList.add(objectSummary.getKey());
            }
            objects = s3.listNextBatchOfObjects(objects);
            System.out.println(objects.isTruncated());
        } while (objects.isTruncated());  //  这里判断是否是可裁剪的，也就是递归地遍历每
个文件夹


        //  将s3的文件同步到本地
        S3ObjectInputStream s3is = null;
        FileOutputStream fos = null;
        try {
            System.out.println("请稍候，正在同步文件...");

            for (String objectPath : objectPathList) {
                //  首先，文件夹不存在的话应该被新建
                if (objectPath.contains("/")) {
                    System.out.println(objectPath);
                    String[] dirs = objectPath.split("/");
                    StringBuilder dir = new StringBuilder(synLocalPath);
                    for (int i = 0; i < dirs.length; ++i) {
                        if (!dirs[i].contains(".")) {
                            dir.append(dirs[i]).append("/");
                        }
                    }
                    File file = new File(dir.toString());
```

```
                    if (!file.exists()) {
                        file.mkdirs();
                    }
                }

                //  处理文件冲突
                // 冲突解决策略：
                // 1. 文件不存在的时候，直接下载；
                // 2. 存在同名文件时，让用户判断是保留本地文件，还是下载云端文件进行覆盖。
                //    --如果选择下载云端文件，那么将会覆盖本地文件；
                //    --如果选择保留本地文件，那么将会将本地文件上传到云端，覆盖云端文件。

                File file = new File(synLocalPath + "/" + objectPath);
                if (!file.exists()) {
                    //  然后开始下载文件夹中的对应文件
                    System.out.println("正在下载：" + objectPath);
                    S3Object o = s3.getObject(bucketName, objectPath);
                    s3is = o.getObjectContent();
                    if (o.getObjectMetadata().getContentLength() > partSize) {
                        download_blocked(s3, bucketName, file, synLocalPath);
                    } else {
                        fos = new FileOutputStream(new File(synLocalPath + "/" +
objectPath));
                        byte[] read_buf = new byte[64 * 1024];
                        int read_len = 0;
                        while ((read_len = s3is.read(read_buf)) > 0) {
                            fos.write(read_buf, 0, read_len);
                        }
                        fos.close();
                        s3is.close();
                    }
                    System.out.println(objectPath + " 下载完成！");

                } else if (!file.isDirectory()) {
                    int choice = 0;
                    System.out.println("存在同名文件 “" + objectPath + "” ，请决定"
+
                            "是保留本地文件还是下载云端文件进行覆盖。");
                    System.out.println("请注意，云端文件和本地文件将会进行同步。");
                    System.out.println("其他数字. 保留本地文件，覆盖云端文件；" +
                            "2. 下载云端文件，覆盖本地文件。");
                    System.out.print("请键入数字进行选择：");
                    choice = scanner.nextInt();
                    if (choice == 2) {
                        System.out.println("本地文件将被覆盖。");
                        System.out.println("正在下载：" + objectPath);
                        S3Object o = s3.getObject(bucketName, objectPath);
                        if (o.getObjectMetadata().getContentLength() > partSize)
{
                            download_blocked(s3, bucketName, file,
synLocalPath);
                        } else {
                            s3is = o.getObjectContent();
                            fos = new FileOutputStream(new File(synLocalPath +
"/" + objectPath));
                            byte[] read_buf = new byte[64 * 1024];
                            int read_len = 0;
                            while ((read_len = s3is.read(read_buf)) > 0) {
```

```
                                fos.write(read_buf, 0, read_len);
                            }
                            fos.close();
                            s3is.close();
                        }
                        System.out.println("已完成本地文件 " + objectPath + " 的覆
盖。");
                    } else {
                        s3.putObject(bucketName, objectPath, file);
                    }
                }
            }

            System.out.println("云端文件本地同步完成~");
        } catch (IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } finally {
            if (s3is != null) try {
                s3is.close();
            } catch (IOException e) {
            }
            if (fos != null) try {
                fos.close();
            } catch (IOException e) {
            }
        }
```

本地文件的添加、修改、删除都同步到云端

该部分处理的主要是在本地文件夹中对文件进行修改之后，通过手动触发程序进行同步。主要是将本地文件同步到云端，其中也包含了对大文件的分块上传。

首先，程序通过设定 while(true) 来实现一个有条件的无限循环，根据用户的输入选择是继续循环还是退出程序。然后，在循环中分别处理用户在本地添加文件、修改文件和删除文件的三种情况。

- 首先是用户添加文件。对于云端不存在的、本地新添加的文件进行上传；而对于云端已经存在的同名文件，将会归在文件修改部分进行处理。同时，对于较大的文件会进行分块上传处理。
- 然后是文件修改部分。第一次启动的时候无法进行文件是否被修改的检测，但会记录下每个文件的最后修改时间；从第二次进行本地文件云端同步的时候，就会使用本次同步获得的文件修改时间来与上次同步记录的文件修改时间进行比较，如不同则视为文件已被修改，上传到云端进行同步。同时，对于较大的文件会进行分块上传处理。
- 最后是文件删除部分。获取云端Bucket的文件目录，检查本地是否存在该文件，若不存在，则在云端删除该文件。

```
        //  接下来将实现本地文件的新建、删除、修改操作同步到云端
        System.out.println("接下来将对本地文件的增删改进行云端同步");
        int flag = 0;
        Map<String, Long> historyFileModifiedTimeList = new HashMap<>();
        List<String> nowFilePathList = null;
        List<String> objectNowList = null;
        while (true) {
            try {
                if (flag == 0) {
```

```
                    System.out.println("请注意，该功能为一有条件无限循环。");
                    flag = 1;
                }
                System.out.println("请选择功能：");
                System.out.println("其他数字. 本地文件云端同步；0. 退出(默认)。");
                int choice = 0;
                choice = scanner.nextInt();
                if (choice == 0) {
                    System.out.println("感谢您的使用！再见！");
                    break;
                } else {
                    try {
                        //  获取此时的本地同步目录
                        nowFilePathList = getFilePathFromDir(synLocalPath);
                        //  获取此时的云端目录
                        objectNowList = new ArrayList<String>();
                        ObjectListing objectsNow = s3.listObjects(bucketName);
                        do {
                            for (S3ObjectSummary objectSummary :
objectsNow.getObjectSummaries()) {
//                                System.out.println("Object: " +
objectSummary.getKey());
                                objectNowList.add(objectSummary.getKey());
                            }
                            objectsNow = s3.listNextBatchOfObjects(objects);
                        } while (objectsNow.isTruncated());
                        //  ↑这里判断是否是可裁剪的，也就是递归地遍历每个文件夹

                        //  这里预计是处理本地添加的文件
                        for (String localFile : nowFilePathList) {
                            if (!objectNowList.contains(localFile)) {
                                File file = new File(synLocalPath + localFile);
                                if (file.exists() && !file.isDirectory()) {
                                    //  遍历本地文件，对云端不存在的、本地新添加的文件进
行上传
                                    //  对于已经存在的文件，可以实现直接覆盖
                                    System.out.println("将本地新添加的文件 " +
localFile + " 进行上传");
                                    if (file.length() > partSize /** 4*/) {
//                                        System.out.println(localFile);
//
System.out.println(file.getCanonicalPath());
                                        System.out.println("文件 " + localFile +
                                                " 较大，将进行分块上传。");
                                        System.out.println("正在分块上传文件 " +
                                                localFile + " ，请稍后...");
                                        upload_blocked(s3, bucketName,
localFile, file);
                                        System.out.println("文件 " + localFile +
" 上传云端成功！");
                                    } else {
                                        System.out.println("正在上传文件 " +
localFile + " ，请稍后...");
                                        s3.putObject(bucketName, localFile,
file);
                                        System.out.println("文件 " + localFile +
" 上传云端成功！");
                                    }
```

```
                                    System.out.println("文件上传" + localFile +
"成功！");

 historyFileModifiedTimeList.put(file.getCanonicalPath(), file.lastModified());
                                }
                            }
                        }


                        //  检测本地文件是否有修改
                        List<File> nowFileList = getFilesFromDir(synLocalPath);
                        if (!historyFileModifiedTimeList.isEmpty()) {
                            System.out.println("开始检测本地文件的修改...");
                            for (File file : nowFileList) {
                                //  异常出现在下一行
//                                  System.out.println(file);
//
System.out.println(historyFileModifiedTimeList.
//                                          get(file.getCanonicalPath()));
                                if (!file.isDirectory() &&
(historyFileModifiedTimeList.
                                        get(file.getCanonicalPath()) !=
file.lastModified())) {
                                    String localPath = file.getCanonicalPath().
                                            replace("\\", "/").
                                            replace(synLocalPath, "");
                                    System.out.println("本地文件 " + localPath + "
已被" +
                                            "修改，进行云端同步...");
                                    if (file.length() > partSize /** 4*/) {
//                                        System.out.println(localFile);
//
System.out.println(file.getCanonicalPath());
                                        System.out.println("文件 " + localPath +
                                                " 较大，将进行分块上传。");
                                        System.out.println("正在分块上传文件 " +
                                                localPath + " ，请稍后...");
                                        upload_blocked(s3, bucketName,
localPath, file);
                                        System.out.println("文件 " + localPath +
" 上传云端成功！");
                                    } else {
                                        System.out.println("正在上传文件 " +
localPath + " ，请稍后...");
                                        s3.putObject(bucketName, localPath,
file);
                                        System.out.println("文件 " + localPath +
" 上传云端成功！");
                                    }
                                    System.out.println("文件 " + localPath + " 云
端同步完成！");
                                    break;
                                }
                            }
                            System.out.println("本地文件的修改检测完成！");
                        } else {
                            System.out.println("本地文件目录无历史信息！");
                        }
```

```
                        for (File file : nowFileList) {
                            //  针对先前已有的key，新value会覆盖之前的value
                            if (!file.isDirectory()) {

 historyFileModifiedTimeList.put(file.getCanonicalPath(),
                                        file.lastModified());
                            }
                        }
                        //  nowFileList.clear();

                        for (File file : nowFileList) {

                        }

                        //  这里预计是处理本地删除的文件
                        //  获取此时的云端文件目录
                        do {
                            for (S3ObjectSummary objectSummary :
objectsNow.getObjectSummaries()) {
//                                System.out.println("Object: " +
objectSummary.getKey());
                                objectNowList.add(objectSummary.getKey());
                            }
                            objectsNow = s3.listNextBatchOfObjects(objects);
                        } while (objectsNow.isTruncated());
                        //  ↑这里判断是否是可裁剪的，也就是递归地遍历每个文件夹

                        nowFilePathList = getFilePathFromDir(synLocalPath);

                        for (String cloudFile : objectNowList) {
                            //  遍历云端文件，检查本地是否存在，不存在则删除
                            if (!nowFilePathList.contains(cloudFile)) {
                                System.out.println("本地不存在 " + cloudFile + " 文
件，正在云端删除...");

                                s3.deleteObject(bucketName, cloudFile);
                                System.out.println("删除云端文件 " + cloudFile + "
成功！");

                            }
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }

            } catch (AmazonClientException e) {
                System.err.println(e.toString());
                System.exit(1);
            } finally {
                if (s3is != null) try {
                    s3is.close();
                } catch (IOException e) {
                }
                if (fos != null) try {
                    fos.close();
                } catch (IOException e) {
                }
            }
```

```
    }
```