**文件同步管理器设计说明书**

软件工程一班 胡剑桥 201830660420

1. **概况**

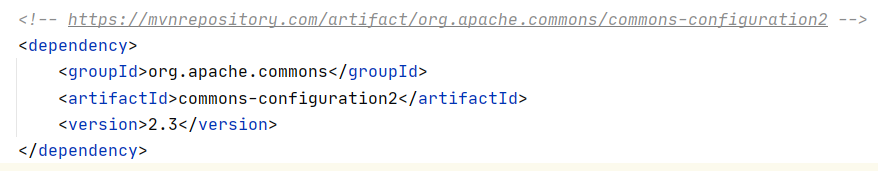
本管理器旨在依托于AWS的S3对象储存服务，将一个个文件以对象的键值对的形式存储在S3数据库里面。本软件实现的功能是使用了AWS给定的Java的SDK将上传与下载的操作进行了封装，使其能够方便用户在选定了本地的一个文件夹之后，程序将自动将本地的文件上传到S3数据库中进行保存，并且，当本地文件有修改或者新增之类的操作之后，程序会监听相应的文件动作，将更新之后的文件保存到云端。对于大文件来说，上传速度可能由于网络问题导致缓慢，本软件提供了分段上传下载的功能，并且在上传的时候如果程序意外退出的话，在软件再一次被打开的时候，会自动继续上一次的传输任务。在软件打开的时候，还会自动检查本地文件与云端文件的版本，如果云端有新的文件或者修改后的文件，软件将会自动下载该文件，同样的，当本地文件中有新的文件或者有某个文件中的较新的版本之后，软件会自动上传该文件至云端。

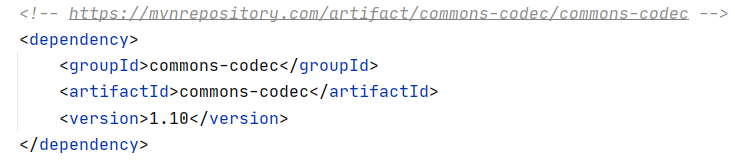
1. **软件安装与使用要求**

系统：本软件使用Java开发，具有一定的跨平台性，开发的系统环境为window10中文家庭版，Intel i5。

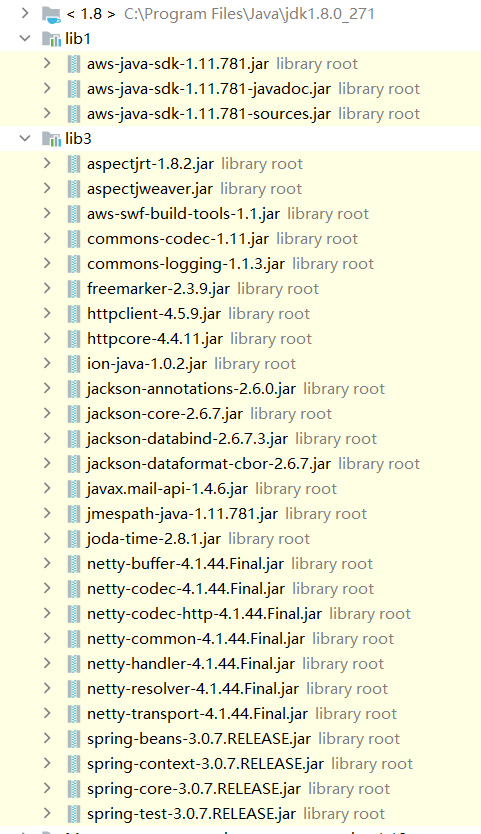
软件要求：jdk1.8，maven，本软件使用了部分由maven管理的jar，具体如下所示：





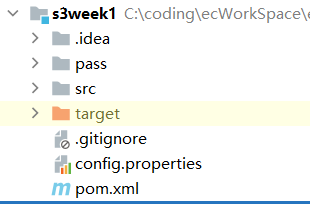


主要由以上三个jar包。并且，项目还应该导入aws的jdk

如下所示：  


1. **主要jar功能分析**
2. junit，主要用于测试
3. org.apache.commons主要用于读取和修改配置文件
4. commons-codec，主要用于计算文件的MD5值，用于和云端对象的ETag值进行对比以确定两个文件是否相同
5. Aws jdk，主要用于连接上传的s3数据库
6. **软件代码概述**

文件整体的代码如下所示：



Pass文件夹里面存放了本软件的使用说明书，PPT等文档。

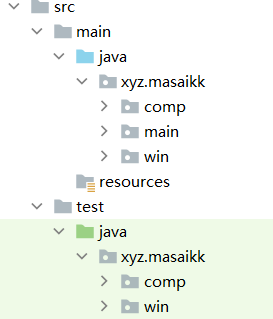
Src里面为主要代码

Target文件夹里面是编译生成的class文件

config.properties文件是本软件的配置文件，具体的内容将在之后进行讨论。

Pom.xml文件是项目maven的配置文件，用于指定需要的maven包。

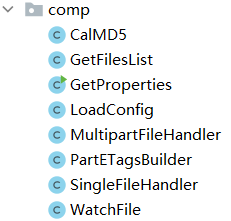
其中，src文件夹内容基本结构如下所示：



main文件夹是软件运行时所需的代码，test文件夹里面的是构建过程中测试的代码。

本项目的包命名为xyz.masaikk.\*

comp包下的是每个功能点的实现类，具体如下所示：



分别对应的功能是：

计算MD5值，获取本文件夹下的文件列表，获取与修改配置文件内容（已弃用），读取配置文件（已弃用），大文件上传下载中断恢复管理，批量PartETag生成（已弃用），多个小型文件上传与下载管理，监视文件列表动作。

main包下面是软件的主类

win包下面是由javafx构建的GUI界面，作为本软件的主体界面，还兼具选择文件夹的功能。

1. **配置文件分析**

代码主目录下的config.properties文件是本软件的配置文件，主要的内容如下所示：



前面两行是由管理配置文件类自动生成的，用来显示上一次修改配置文件的时间

第一行，s3的密钥

第二行，本软件中选定的同步的文件夹的位置

第三行，用于表示当前大文件上传下载的进度，正常情况是0，当出现了上传过程中程序退出的时候，变为1.当出现下载过程中程序退出的时候，变为2

第四行，大文件在进行上传或者下载的操作的时候生成的标识码，可以依次继续上传与下载的过程。

第五行，设置的大文件上传或者下载的分区大小，单位为Mb

第七行，桶的名字

第八行是网址，用于连接S3数据库

第九行，用于标识程序上一次是否正常退出，正常退出即为1，非正常退出则为0

之后几行是大文件在上传与下载过程中每个分区的ETag，行数课随着文件的大小而变化。

之后一行是登录S3的码。

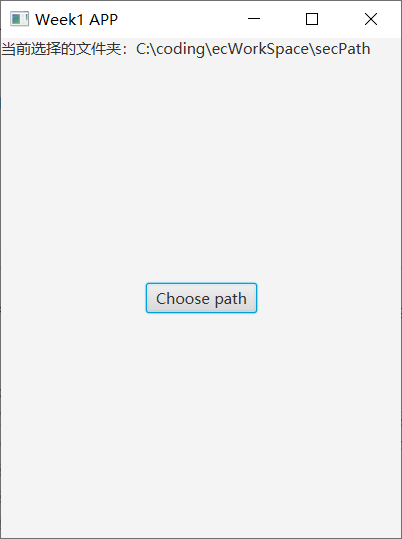
最后一行是正在上传或者下载的大文件的名字。

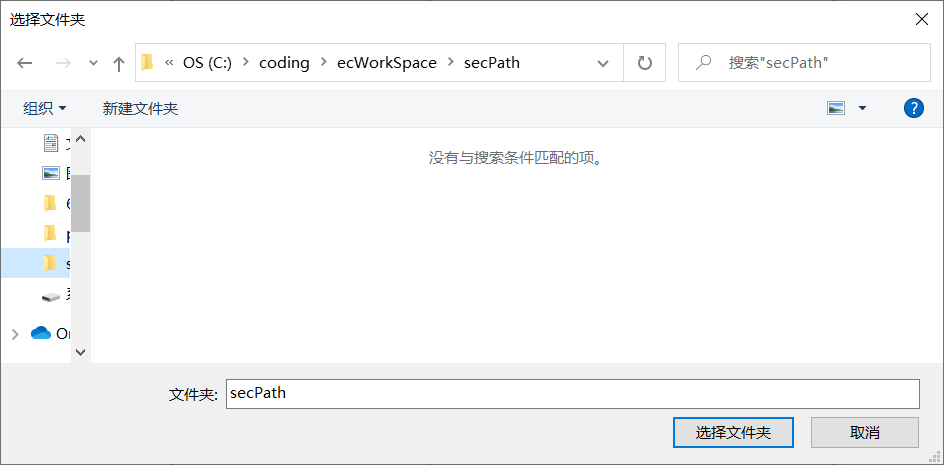
值得注意的是，在软件运行的过程中，由于配置文件的改写，导致行的相对顺序可能会改变。

1. **主要算法及代码分析**
2. 文件夹选择，选择本地的文件夹作为同步的文件夹，使用javafx及其api制作了GUI界面进行文件夹的选择，代码如下所示：



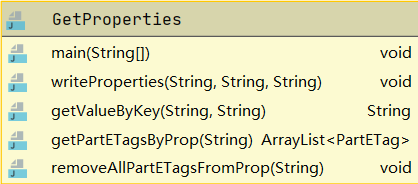
通过选择文件夹，再调用对配置文件修改的类，进行文件的记录。效果为：





选定的文件夹的绝对路径将会记录在配置文件config.properties中，在软件打开的时候会自动读取读取路径，并且会开始接下来的自动同步的过程。

1. 配置文件的读取与更改，为了持久化，本软件使用了properties文件作为软件的配置文件。管理类GetProperties代码如下所示：



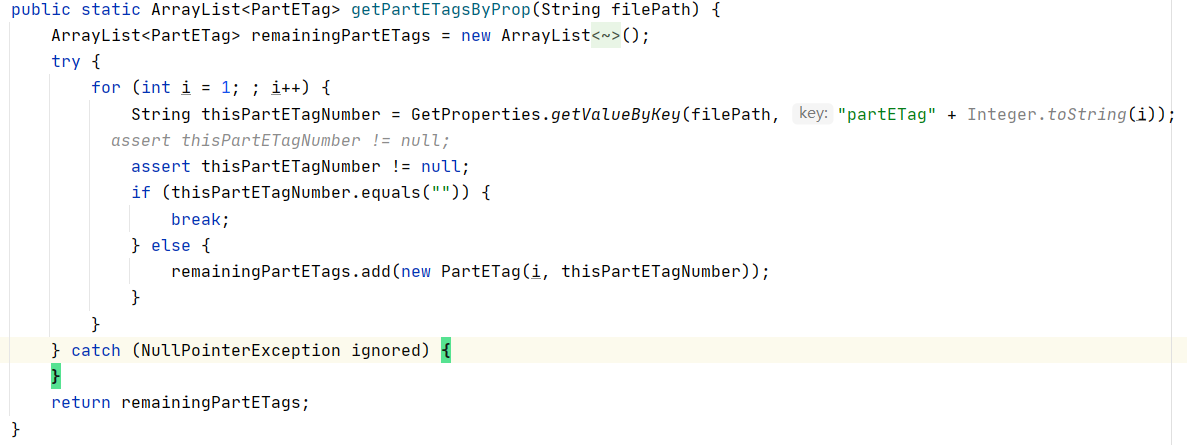
第一个函数已经弃用。

第二个函数用于修改配置数据。

第三个函数用于读取配置数据。

最后两个文件用于调用的便利，在大文件的上传或者下载的过程中，读取生成的临时ETag值以恢复中断的上传或者下载的任务。

例如下图所示

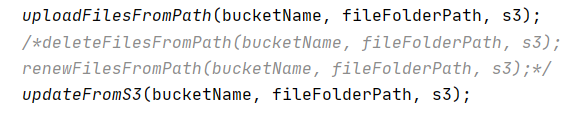


1. 程序开启时自动下载或者上传更新的文件

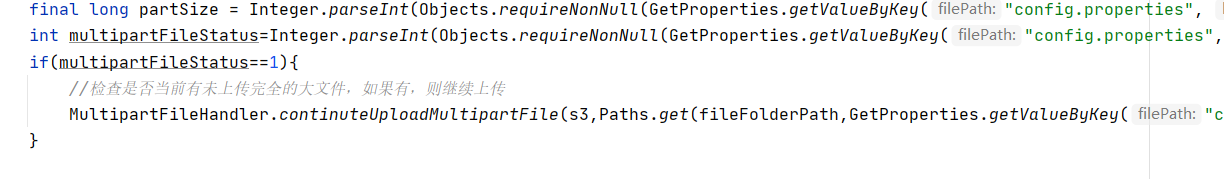
程序在开启的时候会建立s3对象



在之后，先向s3中更新本地新建的文件，再将s3中较新的文件同步到本地

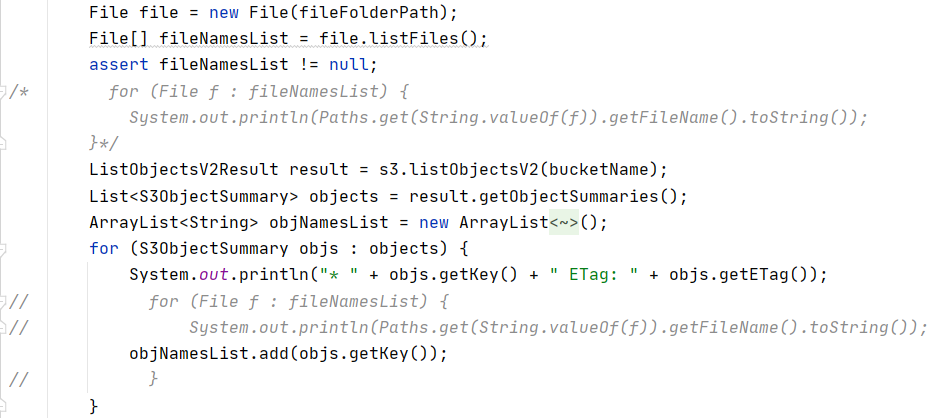


在同步本地所有文件之前，先检查配置文件的multipartFileStatus标识，是否有中断的上传任务：



如果有中断的任务的话，就继续上传，该中断上传的代码实现将在下面详细介绍。

在处理完当前大文件之后，会扫描文件夹，获取当前文件的列表，并且读取s3中的各个文件列表：



对比两个列表，使用ArrayList.contains()函数，如果本地存在一个s3上没有的文件，就自动上传，值得注意的是，这里需要比较文件的大小，如超过了配置文件中的part大小，则使用大文件上传的函数将其分段上传。

对于小文件来说，就直接使用函数

s3.putObject(bucketName, Paths.*get*(String.*valueOf*(f)).getFileName().toString(), new File(String.*valueOf*(f)));

将其上传。

1. 文件冲突的管理

有些文件在s3和本地均存在，但是内容不同，一个保留更加新的文件。

当存在同名的文件的时候，先提取s3文件的Etag值，即该文件的MD5值。再计算本地文件的MD5值，使用如下函数：

static public String calculate(Path path) {  
 try (InputStream is = Files.*newInputStream*(Paths.*get*(String.*valueOf*(path)))) {  
 *//calculate md5 and return it.* return org.apache.commons.codec.digest.DigestUtils.*md5Hex*(is);  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "";  
 }  
}

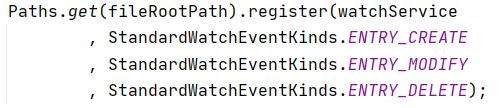
先对比两个md5值，如果相同，则这两个文件是相同的，则不需要做任何操作。如果不同，则必须要做一个取舍。此时对比本地文件和s3文件的修改时间，保留时间戳更大文件。

即可解决文件冲突。

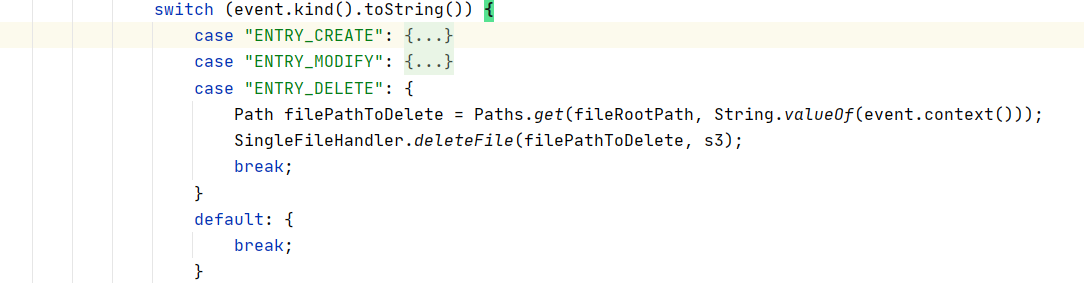
1. 文件操作监听

使用了类java.nio.file.\*对文件进行监听，在没有文件操作的时候，阻塞。

定义了三个文件动作，分别对应与文件的创建，修改和删除。



在选定的文件夹中有相关操作时，便触发对应函数，在此也应该比较文件大小，是否分段上传。

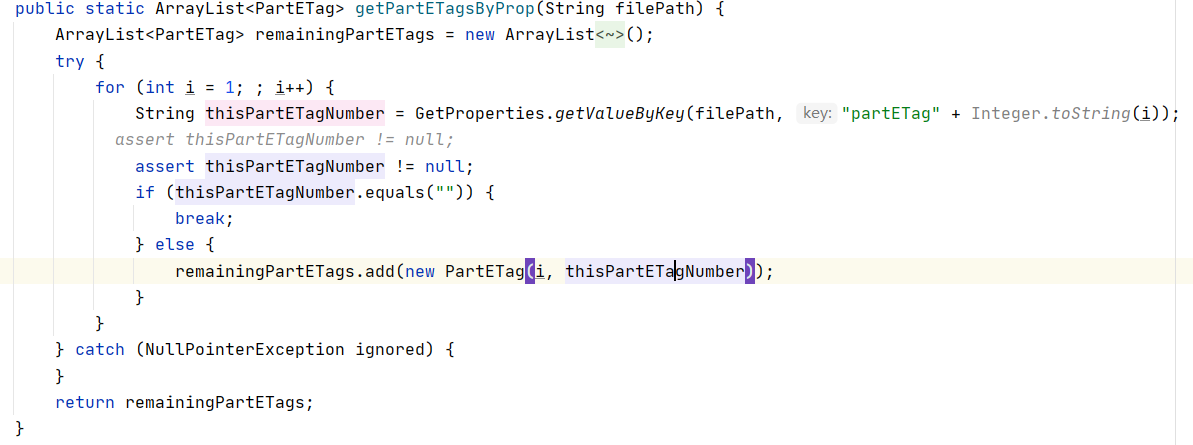


1. 大文件上传及下载中断和恢复功能

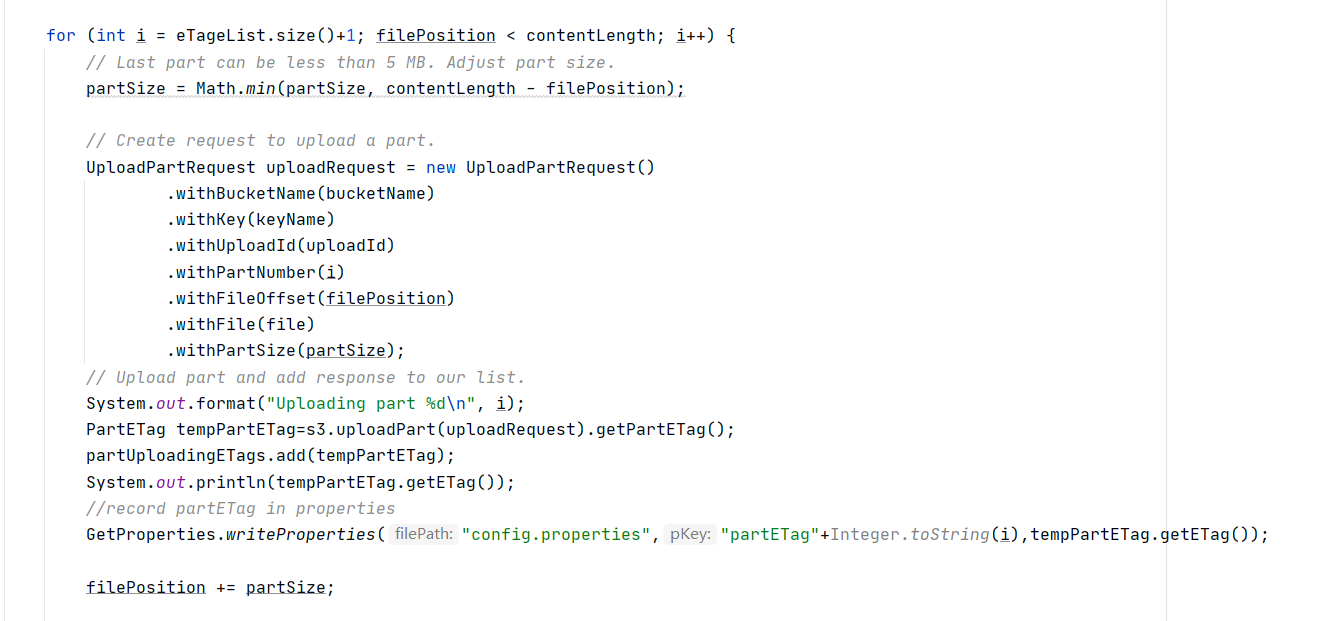
在一次传输请求的时候，对于同一个大文件来说，生成的每一个上传的请求都有一个独立的码，在每个大文件上传或者下载的时候，先改变配置文件中的值multipartFileStatus，在这个上传或者下载的功能完全结束之后，将这个multipartFileStatus改回0.表示这个过程结束。在修改过程中记录下大文件的这个记录值，在配置文件的multipartFileID部分，根据这个独立的值，可以确定一次大文件的传输过程。在文件传输的过程中，每一个Part都会生成独立的ETag值，在每一个part传输过程中，依次记录下这些PartETag值，并且将它们持久化保存，如下所示：



在传输过程中，软件如果意外中断的话，这些已经传输成功的Etag值还是能保存在配置文件中。在启动程序的时候，检查配置文件中的multipartFileStatus，确定还有一个传输的过程并未正常完成，所以读取配置文件中的全部PartETag值，并且生成PartETag数组，如下所示：

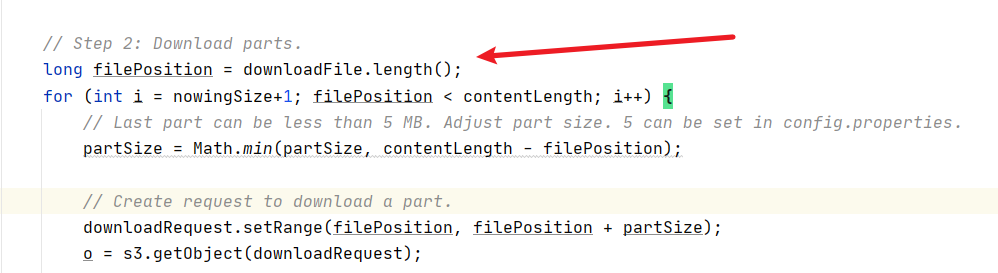


而根据生成的PartEtag数组的长度，可以确定当前传输任务应该开始的程度，并且根据之前记录的multipartFileID值，完成剩余的传输任务，如下图所示。

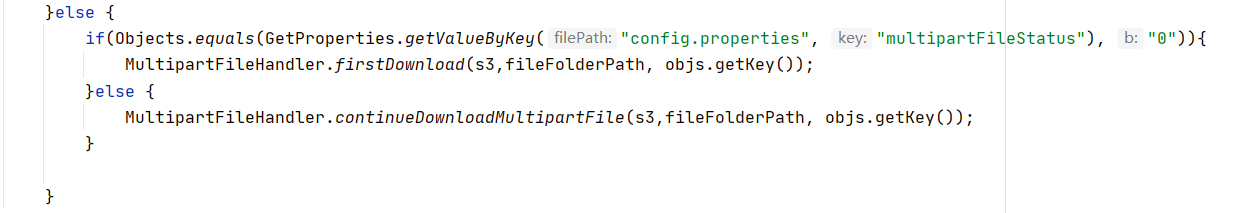


在完成了一个完整的传输任务之后，会将multipartFileStatus位设置为0，标志着传输任务的正式完成。大文件的下载分段与恢复的具体实现与上传的分段恢复类似，在此便不再赘述，但是值得注意的是，下载的时候还应该用本地现存的文件读取已经下载文件的大小来确定下载的进度，之后的可以通过文件大小的划分来确定下载的进度。

如下图所示：



在下载的时候开始一个文件status的判断即可



以上是软件的设计说明。