实验目标:

1. 编程实现本地文件夹与Bucket的文件同步功能

需求：

功能性需求

1. 远端文件同步到本地：程序启动是从Bucket中拉去文件到本地, 并进行文件的冲突处理

2. 本地文件同步到远端：本地的文件变化更新到Bucket, 如文件修改, 删除, 添加.

非功能性需求：

3. 当文件超过20MB时, 使用分块上传

4. 大文件传输中断时，选择性重传缺失块

5. 使用多线程分块上传、下载大文件

6 为变化的文件不需要重新传输

概要设计:

程序提供两个事务类Uploader和Downloader， 一个控制函数main

Uploader负责本地文件同步到远端事务, 内部实现upload\_folder()函数用于扶着同步逻辑；delete\_exfiles()负责删除远端文件；uploadfiles()负责上传文件到远端；multipart\_upload()实现大文件分块上传；recover\_upload()用于中断续传。

Download负责远端文件同步到本地事务，内部实现： download\_file ()作为内部下载的逻辑控制；multipartdownload()实现分块下载；recover\_downloaded()实现断点续传。

实现简介:

1. 程序启动是，根据config.txt的配置信息，建立本地文件和远端桶的连接关系，并初始化

Uploader和Downloader事务类。

程序提供两个简约指令download和upload，用于启动远端文件同步到本地和本地文件同步到远端事务，整体的大文件分块传输和断点续传内部制动处理。

Download介绍：

用s3\_client.list\_objects(Bucket=bucket\_name)获取远端的所有文件的元数据信息，

对比每个文件的本地文件的状态情况，决定是否需要下载

对于下载的文件大于20M的情况，启动分块下载，并启用多线程

每个分块下载前，先查询本地是否有缓存，对比信息后，决定是否重新下载

处理所有分块后检查，本地缓存文件的summary信息，查看是否所有分块就位。

执行merge\_multipart()，将本地缓存分块合并为源文件。

Uploader介绍：

用get\_all\_url获取本地的所有文件的元数据信息，

对比每个文件的远程文件的状态情况，决定是否需要上传，或者删除远程文件

对于上传的文件大于20M的情况，启动分块上传，并启用多线程

每个分块上传前，先查询远端是否有缓存，对比信息后，决定是否重新上传

处理所有分块前检查，远端缓存文件的状态信息，重传缺失分块。

执行complete\_multipart\_upload，将远程缓存分块合并为源文件。

重点的功能点思想：

下载断点续传的过程大致如下：  
1. 在本地创建一个临时文件，文件名由原始文件名加上一个随机的后缀组成；  
2. 通过指定HTTP请求的 Range 头，按照范围读取对象存储服务器上的文件，并写入到临时文件里相应的位置；  
3. 下载完成之后，把临时文件重名为目标文件。

分片上传可以分为三个步骤：  
1. 初始化（createMultipartUpload）：获得Upload ID  
2. 上传分片（uploadPart）：这一步可以并发进行  
3. 完成上传（completeMultipartUpload）：合并分片，生成文件