设计说明书

程序功能

指定本地某个目录与s3的某个Bucket进行文件同步。

程序要求

（1）程序启动时把Bucket中的文件同步到本地。需要处理文件冲突

（2）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。

（3）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。

程序配置

系统环境：window10

开发环境：java 1.8.0\_251

开发支持：Apache Commons工具包，AWS-SDK-java库

程序说明

1. AWS\_S3模块

类名：S3

文件名称：S3.java

功能：S3服务器操作

类成员变量：

1、bucketName

描述：bucket名称。

属性：私有

类型：String。

2、savePath;

描述：本地同步路径。

属性：私有

类型：String。

3、accessKey;

描述：S3访问识别码。

属性：私有

类型：String。

4、secretKey;

描述：S3访问密钥。

属性：私有

类型：String。

5、serviceEndpoint;

描述：S3服务器地址。

属性：私有

类型：String。

6、signingRegion = "";

描述：S3访问规则。

属性：私有

类型：静态String。

7、s3;

描述：S3服务器对象。

属性：私有

类型：AmazonS3。

类成员函数：

1. S3(String, String, String, String, String)

描述：构造函数，构建S3对象并进行初始化。

参数：String BucketName, String SavePath, String AccessKey, String SecretKey, String ServiceEndpoint，分别是bucket名称、保存路径、访问令牌、访问密钥和服务器地址。

操作：传入参数，对成员变量进行初始化操作。然后根据bucket名称、保存路径、访问令牌、访问密钥和服务器地址等信息连接S3服务器，建立s3对象。

返回值：无

详细代码：

**public** S3(String BucketName, String SavePath, String AccessKey, String SecretKey, String ServiceEndpoint) {

bucketName = BucketName;

savePath = SavePath;

accessKey = AccessKey;

secretKey = SecretKey;

serviceEndpoint = ServiceEndpoint;

**final** BasicAWSCredentials credentials = **new** BasicAWSCredentials(accessKey,secretKey);

**final** ClientConfiguration ccfg = **new** ClientConfiguration().withUseExpectContinue(**true**);

**final** EndpointConfiguration endpoint = **new** EndpointConfiguration(serviceEndpoint, ***signingRegion***);

s3 = AmazonS3ClientBuilder.*standard*()

.withCredentials(**new** AWSStaticCredentialsProvider(credentials))

.withClientConfiguration(ccfg)

.withEndpointConfiguration(endpoint)

.withPathStyleAccessEnabled(**true**)

.build();

}

1. delete(String)

描述：根据本地删除的文件，删除S3服务器中指定文件。

参数：String filePath表示的是本地删除文件的路径。

操作：先对本地删除文件的地址进行路径，根据地址从S3中的文件夹中寻找指定文件并将其删除。

返回值：无

详细代码：

**public** **void** delete(String filePath) {

String keyName = Paths.*get*(filePath).toString().replace(savePath+"\\", "").replace("\\", "/");

**try** {

s3.deleteObject(bucketName, keyName);

} **catch** (AmazonServiceException e) {

System.***err***.println(e.getErrorMessage());

System.*exit*(1);

}

System.***out***.printf("删除 %s 文件成功!\n", keyName);

}

1. upload(String)

描述：根据本地添加的文件，往S3服务器上传中指定文件。

参数：String filePath表示的是本地删除文件的地址。

操作：先对本地添加文件的地址进行处理，根据地址确定要往S3中那个文件夹上传。然后对本地文件进行分块，以20Mb为一块分段上传。

返回值：无

详细代码：

**public** **void** upload(String filePath) {

String keyName = Paths.*get*(filePath).toString().replace(savePath+"\\", "").replace("\\", "/");

// Create a list of UploadPartResponse objects. You get one of these

// for each part upload.

ArrayList<PartETag> partETags = **new** ArrayList<PartETag>();

File file = **new** File(filePath);

**long** contentLength = file.length();

String uploadId = **null**;

**try** {

// Step 1: Initialize.

InitiateMultipartUploadRequest initRequest =

**new** InitiateMultipartUploadRequest(bucketName, keyName);

uploadId = s3.initiateMultipartUpload(initRequest).getUploadId();

System.***out***.format("Created upload ID was %s\n", uploadId);

// Step 2: Upload parts.

**long** filePosition = 0;

**long** partSize = 20 << 20;

**for** (**int** i = 1; filePosition < contentLength; i++) {

// Last part can be less than 20 MB. Adjust part size.

partSize = Math.*min*(partSize, contentLength - filePosition);

// Create request to upload a part.

UploadPartRequest uploadRequest = **new** UploadPartRequest()

.withBucketName(bucketName)

.withKey(keyName)

.withUploadId(uploadId)

.withPartNumber(i)

.withFileOffset(filePosition)

.withFile(file)

.withPartSize(partSize);

// Upload part and add response to our list.

System.***out***.format("Uploading part %d\n", i);

partETags.add(s3.uploadPart(uploadRequest).getPartETag());

filePosition += partSize;

}

// Step 3: Complete.

CompleteMultipartUploadRequest compRequest =

**new** CompleteMultipartUploadRequest(bucketName, keyName, uploadId, partETags);

s3.completeMultipartUpload(compRequest);

System.***out***.printf("Completing upload %s\n", keyName);

}

**catch** (Exception e) {

System.***err***.println(e.toString());

**if** (uploadId != **null** && !uploadId.isEmpty()) {

// Cancel when error occurred

System.***out***.println("Aborting upload");

s3.abortMultipartUpload(**new** AbortMultipartUploadRequest(bucketName, keyName, uploadId));

}

System.*exit*(1);

}

//System.out.printf("上传 %s 文件成功!\n", keyName);

}

1. synchronize (String)

描述：根据本地添加的文件，往S3服务器上传中指定文件。

参数：String filePath表示的是本地删除文件的地址。

操作：先对本地添加文件的地址进行处理，根据地址确定要往S3中那个文件夹上传。然后对本地文件进行分块，以20Mb为一块分段上传。

返回值：无

详细代码：

**public** **void** synchronize(){

**final** BasicAWSCredentials credentials = **new** BasicAWSCredentials(accessKey,secretKey);

**final** ClientConfiguration ccfg = **new** ClientConfiguration().

withUseExpectContinue(**true**);

**final** EndpointConfiguration endpoint = **new** EndpointConfiguration(serviceEndpoint, ***signingRegion***);

**final** AmazonS3 s3 = AmazonS3ClientBuilder.*standard*()

.withCredentials(**new** AWSStaticCredentialsProvider(credentials))

.withClientConfiguration(ccfg)

.withEndpointConfiguration(endpoint)

.withPathStyleAccessEnabled(**true**)

.build();

ListObjectsV2Result result = s3.listObjectsV2(bucketName);

List<S3ObjectSummary> objects = result.getObjectSummaries();

**for** (S3ObjectSummary os : objects) {

String cnt\_File = os.getKey();

**final** String filePath = Paths.*get*(savePath, cnt\_File).toString();

File file = **new** File(filePath);

**if** (cnt\_File.substring(cnt\_File.length()-1).equals("/")) {

//File file = new File(filePath);

**if**(!file.exists()){//如果文件夹不存在

file.mkdirs();//创建文件夹

}

**continue**;

}

File pofile = **new** File(filePath.substring(0, filePath.lastIndexOf("\\")));

**if**(!pofile.exists()){//如果文件夹不存在

pofile.mkdirs();//创建文件夹

}

S3Object o = **null**;

S3ObjectInputStream s3is = **null**;

FileOutputStream fos = **null**;

String keyName = cnt\_File;

**try** {

// Step 1: Initialize.

StringBuilder s = **new** StringBuilder(file.getPath());

s.insert(s.lastIndexOf("\\")+1, "(old)");

File change\_file = file;

File changed\_file = **new** File(**new** String (s));

**if** (change\_file.renameTo(changed\_file)) {

System.***out***.printf("文件 %s 冲突，已修改原文件名\n", file.getPath());

}

ObjectMetadata oMetaData = s3.getObjectMetadata(bucketName, keyName);

**final** **long** contentLength = oMetaData.getContentLength();

**final** GetObjectRequest downloadRequest = **new** GetObjectRequest(bucketName, keyName);

fos = **new** FileOutputStream(file);

// Step 2: Download parts.

**long** filePosition = 0;

**long** partSize = 20 << 20;

**for** (**int** i = 1; filePosition < contentLength; i++) {

// Last part can be less than 20 MB. Adjust part size.

partSize = Math.*min*(partSize, contentLength - filePosition);

// Create request to download a part.

downloadRequest.setRange(filePosition, filePosition + partSize);

o = s3.getObject(downloadRequest);

// download part and save to local file.

System.***out***.format("Downloading part %d\n", i);

filePosition += partSize+1;

s3is = o.getObjectContent();

**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);

}

}

// Step 3: Complete.

System.***out***.println("Completing download");

System.***out***.format("save %s to %s\n", keyName, filePath);

} **catch** (Exception 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) { }

}

System.***out***.printf("同步 %s 文件完成", keyName);

}

System.***out***.println("同步完成!");

}

1. 事件激活模块

类名：FileListener

文件名称：FileListener.java

功能：对监控器产生的文件修改信息做出相应。

描述：该类继承了commons.io.monitor.FileAlterationListenerAdaptor类，在本程序中对其相应的事件响应函数进行重载来实现同步的功能。

类成员变量：

1、aws\_S3

描述：亚马逊S3服务器对象。

属性：私有

类型：S3。

程序主函数设计思路

首先初始化一个aws\_S3服务器对象并对服务器进行连接。然后创建一个文件监听器对指定的目录进行监听，接着重载监听器中响应的激活事件（例如监听到新增文件就上传到服务器）。最后启动监听器完成目标。