# MinIO 断点续传、分片上传

[参考此项目](http://cxyroad.com/”https://gitee.com/Gary2016/minio-upload”)

此项目是 vue+spring boot ，我来介绍他的代码逻辑。看他如何做分片上传和断点续传的。

我们以前端页面为入口。他是用 el-upload 组件进行上传，handleHttpRequest方法。

## 获取任务信息

对于上传一个大文件来说，每个上传大文件都是一个任务，这个任务在数据库的结构为：

```
@Data
@Accessors(chain = true)
@TableName(”sys_upload_task”)
public class SysUploadTask implements Serializable {

    private Long id;
    //分片上传的uploadId
    private String uploadId;
    //文件唯一标识（md5）
    private String fileIdentifier;
    //文件名
    private String fileName;
    //所属桶名
    private String bucketName;
    //文件的key
    private String objectKey;
    //文件大小（byte）
    private Long totalSize;
    //每个分片大小（byte）
    private Long chunkSize;
```

```
    //分片数量
    private Integer chunkNum;

}
```

为了给每个文件做唯一标识，他在前端页面调用 sparkMD5 这个库计算大文件的 md5 值。

使用 md5 值去调用后端接口，查看此任务相关信息。如果这个上传任务已经被创建，那么查看此任务的已经被上传的部分，返回已被上传得分片信息。如果上传任务还没有被创建，那么则初始化任务。

初始化任务
---

```
InitiateMultipartUploadResult initiateMultipartUploadResult = amazonS3
        .initiateMultipartUpload(new
InitiateMultipartUploadRequest(bucketName,
key).withObjectMetadata(objectMetadata));
String uploadId = initiateMultipartUploadResult.getUploadId();
```

告诉 minio 我们要上传的 obejct name，之后我们获得一个 uploadId。

将任务信息存到数据库，比如任务文件名，objectkey，uploadId，md5，文件总大小，分片大小，总共分了多少片。

上传文件分片
---

一个大文件是分片的了，在我们调用获取任务信息时，就向 minio 调用接口去查询已经上传的 part，minio 会返回一个 list，这个 list 里面会存放已经上传完成的 part。使用 for 循环时，先判断此 part 是否已经上传，如果已经上传则略过更新 UI。如果没有上传，则前端将 file 进行分片，然后想后端调用接口获得上传分片的预签名url.

```
params.put(”partNumber”, partNumber.toString());
params.put(”uploadId”, task.getUploadId());

@Override
public String genPreSignUploadUrl(String bucket, String objectKey,
Map<String, String> params) {
    Date currentDate = new Date();
    Date expireDate = DateUtil.offsetMillisecond(currentDate,
MinioConstant.PRE_SIGN_URL_EXPIRE.intValue());
    GeneratePresignedUrlRequest request = new
GeneratePresignedUrlRequest(bucket, objectKey)
            .withExpiration(expireDate).withMethod(HttpMethod.PUT);
    if (params != null) {
        params.forEach((key, val) -> request.addRequestParameter(key,
val));
    }
    URL preSignedUrl = amazonS3.generatePresignedUrl(request);
    return preSignedUrl.toString();
}
```

那么这样 minio 就知道了上传的一个分片的文件对应的 partNumber。

前端拿到 url 后，根据这个 url 上传分片的文件。

> BUG：源码中关于已经上传的 part 的校验，仅仅只是校验了 partNum 是否存在，还应该校验每个 partsize 是否与分片对应的 partsize 一样，如果一样才能说明真的这个分片传输完成了。因为有可能某个分片只传输了一半就暂停了。

合并文件
----

将每个文件分片上传成功后，调用 minio的 CompleteMultipartUploadRequest接口，进行分片的合并。

```
    @Override
```

```
    public void merge(String identifier) {
        SysUploadTask task = getByIdentifier(identifier);
        if (task == null) {
            throw new RuntimeException("分片任务不存");
        }

        ListPartsRequest listPartsRequest = new
ListPartsRequest(task.getBucketName(), task.getObjectKey(),
task.getUploadId());
        PartListing partListing = amazonS3.listParts(listPartsRequest);
        List<PartSummary> parts = partListing.getParts();
        if (!task.getChunkNum().equals(parts.size())) {
            // 已上传分块数量与记录中的数量不对应，不能合并分块
            throw new RuntimeException("分片缺失，请重新上传");
        }
        CompleteMultipartUploadRequest completeMultipartUploadRequest
= new CompleteMultipartUploadRequest()
                .withUploadId(task.getUploadId())
                .withKey(task.getObjectKey())
                .withBucketName(task.getBucketName())
                .withPartETags(parts.stream().map(partSummary -> new
PartETag(partSummary.getPartNumber(),
partSummary.getETag())).collect(Collectors.toList()));
        CompleteMultipartUploadResult result =
amazonS3.completeMultipartUpload(completeMultipartUploadRequest);
    }

```

对于 parts 的校验有问题，仅仅是校验了 parts 的数量，还应该校验 part 的 size。

原文链接: https://juejin.cn/post/7381476230775701558