劝退记录：大文件上传

需求

上传大文件到minio

分析

查询资料有方案如下：

一、切片上传

前端将大文件进行分片，例如一个50MB的文件，分成10片，每个片5MB。然后发10个HTTP请求，将这10个分片数据发送给后端，后端根据分片的下标和Size来往磁盘文件的不同位置写入分片数据，10个分片全部写完后即得到一个完整的文件。

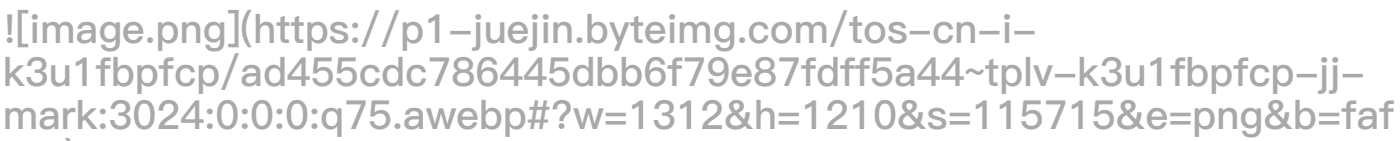
二、ossutil

ossutil支持通过Windows、Linux和[macOS](http://cxyroad.com/”https://so.csdn.net/so/search?q=macOS&spm=1001.2101.3001.7020”)系统以命令行方式管理OSS数据。

使用ossutil的话就需要考虑如何集成到后端系统，如何监控的问题，故最终选择切片上传

实现

流程设计



接口划分

1.获取上传文件接口

```
/**
 * 获取上传文件
 */
@ApiLog(value = "获取上传文件")
@ApiOperation(value = "获取上传文件")
@GetMapping("/getUploadingFile/{fileMD5}")
@ApiOperationSupport(order = 1)
public R getUploadingFile(@PathVariable String fileMD5) {
    if (StringUtils.isBlank(fileMD5)) {
        throw new ServiceException("未传文件MD5值");
    }
    HashMap<String, Object> redisMap = (HashMap<String,
Object>)redisTemplate.opsForHash().entries(fileMD5);
    if (redisMap != null && redisMap.size() > 0) {
        FileUploadInfo fileUploadInfo = getFileUploadInfo(redisMap);
        // 查询上传后的分片数据
        MinioClient minioClient = ossBuilder.minioClient();
fileUploadInfo.setChunkUploadedList(MinioUtils.getChunkByFileMD5(mini
oClient, fileUploadInfo.getFileName(), fileUploadInfo.getUploadId(), "oa-
cloud"));
        return R.data(fileUploadInfo);
    }
    return R.fail("未找到上传文件");
```

}

```

2.检验文件是否存在

```
/**
 * 校验文件是否存在
 *
 * @param md5 String
 * @return ResponseResult<Object>
 */
@GetMapping("/multipart/check")
@ApiOperation(value = "校验文件是否存在")
@ApiOperationSupport(order = 2)
public R checkFileUploadedByMd5(@RequestParam("md5") String md5)
{
    if (StringUtils.isEmpty(md5)) {
        throw new ServiceException("访问参数无效");
    }
    return getByFileMD5(md5);
}
```

3.分片初始化

```
/**
 * 分片初始化
 *
 * @param fileUploadInfo 文件信息
 * @return ResponseResult<Object>
 */
@PostMapping("/multipart/init")
@ApiOperation(value = "分片初始化")
@ApiOperationSupport(order = 3)
public R initMultiPartUpload(@RequestBody FileUploadInfo
fileUploadInfo) {
    return R.data(init(fileUploadInfo));
}
```
```

4.完成上传

```
/**
 * 完成上传
 *
 * @param fileUploadInfo 文件信息
 * @return ResponseResult<Object>
 */
@PostMapping("/multipart/merge")
@ApiOperation(value = "完成上传")
@ApiOperationSupport(order = 4)
public R completeMultiPartUpload(@RequestBody FileUploadInfo
fileUploadInfo) {
    //合并文件
    String url = mergeMultipartUpload(fileUploadInfo);
    //获取上传文件地址
    if (StringUtils.isNotBlank(url)) {
        return R.data(url);
    }
    throw new ServiceException("上传失败");
}
```

5.上传文件

```
@PostMapping("/multipart/uploadScreenshot")
@ApiOperation(value = "上传切片文件")
@ApiOperationSupport(order = 5)
@SneakyThrows
public R uploaduploadScreenshot(@RequestPart("photos") MultipartFile[]
photos,
                                @RequestParam("buckName") String buckName) {

    for (MultipartFile photo : photos) {
        if (!photo.isEmpty()) {
            BladeFile bladeFile =
ossBuilder.template().putFile(photo.getOriginalFilename(),
photo.getInputStream());
        }
    }
```

```
    return R.status(true);
}
```

6.提出来的service方法

```
public String mergeMultipartUpload(FileUploadInfo fileUploadInfo) {
    HashMap<String, Object> redisMap = (HashMap<String,
Object>)redisTemplate.opsForHash().entries(fileUploadInfo.getFileMd5());
    FileUploadInfo redisFileUploadInfo = getFileUploadInfo(redisMap);
    if (redisFileUploadInfo != null) {
        fileUploadInfo.setFileName(redisFileUploadInfo.getFileName());
    }
    MinioClient minioClient = ossBuilder.minioClient();
    boolean result = MinioUtils.mergeMultipartUpload(minioClient,
fileUploadInfo.getFileName(), fileUploadInfo.getUploadId(), "oa-cloud");
    //合并成功
    if (result) {
        //存入数据库
        ShotFile files = saveMergeFileToDB(fileUploadInfo);
        redisTemplate.delete(fileUploadInfo.getFileMd5());
        return files.getFileUrl();
    }
    return null;
}

private ShotFile saveMergeFileToDB(FileUploadInfo fileUploadInfo) {
    String suffix =
fileUploadInfo.getFileName().substring(fileUploadInfo.getFileName().lastIn
dexOf("."));
    Date date = DateUtil.date();
    String today = DateUtil.format(date, "yyyyMMdd");
    String url = oss + "/" + fileUploadInfo.getFileName();
    //存入数据库
    ShotFile files = new ShotFile();
    BeanUtils.copyProperties(fileUploadInfo, files);
    files.setFileUrl(url);
    shotFileMapper.insert(files);
    return files;
}
```

```
private ShotFile saveFileToDB(FileUploadInfo fileUploadInfo) {
    String suffix =
fileUploadInfo.getFileName().substring(fileUploadInfo.getFileName().lastIn
dexOf(”.”));
    Date date = DateUtil.date();
    String today = DateUtil.format(date, ”yyyyMMdd”);
    String url = oss + shot_path + ”/” + today + ”/”
+fileUploadInfo.getFileMd5() + suffix;
    //存入数据库
    ShotFile files = new ShotFile();
    BeanUtils.copyProperties(fileUploadInfo, files);
    files.setFileUrl(url);
    shotFileMapper.insert(files);
    return files;
}

public R getByFileMD5(String md5) {
    // 从redis获取文件名称和id
    HashMap<String, Object> redisMap = (HashMap<String,
Object>)redisTemplate.opsForHash().entries(md5);
    FileUploadInfo fileUploadInfo = getFileUploadInfo(redisMap);
    if (fileUploadInfo != null) {
        // 正在上传，查询上传后的分片数据
        MinioClient minioClient = ossBuilder.minioClient();
        List<Integer> chunkList =
MinioUtils.getChunkByFileMD5(minioClient, fileUploadInfo.getFileName(),
fileUploadInfo.getUploadId(), ”oa-cloud”);
        fileUploadInfo.setChunkUploadedList(chunkList);
        //正在上传
        fileUploadInfo.setStatus(2);
        return R.data(fileUploadInfo);
    }
    // 查询数据库是否上传成功
    ShotFile one = shotFileMapper.selectOne(new
LambdaQueryWrapper<ShotFile>().eq(ShotFile::getFileMd5, md5));
    if (one != null) {
        FileUploadInfo mysqlsFileUploadInfo = new FileUploadInfo();
        BeanUtils.copyProperties(one, mysqlsFileUploadInfo);
        mysqlsFileUploadInfo.setStatus(1);
        return R.data(mysqlsFileUploadInfo);
    }
    FileUploadInfo result = new FileUploadInfo();
    result.setStatus(0);
    return R.data(result);
}

```

```
public Map<String, Object> init(FileUploadInfo fileUploadInfo) {
    HashMap<String, Object> redisGetMap = (HashMap<String,
Object>)redisTemplate.opsForHash().entries(fileUploadInfo.getFileMd5());
    FileUploadInfo redisFileUploadInfo = getFileUploadInfo(redisGetMap);
    if (redisFileUploadInfo != null) {
        fileUploadInfo = redisFileUploadInfo;
    }
    // 单文件上传
    if (fileUploadInfo.getChunkNum() == 1) {
        //保存文件
        ShotFile files = saveFileToDB(fileUploadInfo);
        String fileName =
files.getFileUrl().substring(files.getFileUrl().lastIndexOf("/") + 1);
        return MinioUtils.getUploadObjectUrl(files.getFileUrl());
    }
    // 分片上传
    else {
        MinioClient minioClient = ossBuilder.minioClient();
        Map<String, Object> map =
MinioUtils.initMultiPartUpload(minioClient, fileUploadInfo,
fileUploadInfo.getFileName(), fileUploadInfo.getChunkNum(),
fileUploadInfo.getContentType(), "oa-cloud");
        String uploadId = (String) map.get("uploadId");
        fileUploadInfo.setUploadId(uploadId);
        HashMap<String, Object> redisMap = new HashMap<>();
        redisMap.put("fileName",fileUploadInfo.getFileName());
        redisMap.put("fileSize",fileUploadInfo.getFileSize());
        redisMap.put("contentType",fileUploadInfo.getContentType());
        redisMap.put("chunkNum",fileUploadInfo.getChunkNum());
        redisMap.put("uploadId",fileUploadInfo.getUploadId());
        redisMap.put("chunkSize",fileUploadInfo.getChunkSize());
        redisMap.put("fileMd5",fileUploadInfo.getFileMd5());
        redisMap.put("fileType",fileUploadInfo.getFileType());
redisMap.put("chunkUploadedList",fileUploadInfo.getChunkUploadedList
());
redisTemplate.opsForHash().putAll(fileUploadInfo.getFileMd5(),redisMap);

        redisTemplate.expire(fileUploadInfo.getFileMd5(),
Duration.ofDays(1));
        return map;
    }
}

```

7.MinioUtil

```
package org.springblade.resource.utils;

import cn.hutool.core.text.CharSequenceUtil;
import com.google.common.collect.HashMultimap;
import io.minio.GetPresignedObjectUrlArgs;
import io.minio.ListPartsResponse;
import io.minio.MinioClient;
import io.minio.http.Method;
import io.minio.messages.Part;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springblade.core.log.exception.ServiceException;
import org.springblade.core.tool.api.R;
import org.springblade.resource.entity.FileUploadInfo;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;


@Slf4j
public class MinioUtils {

    private CustomMinioClient customMinioClient;

    /**
     * 通过 sha256 获取上传中的分片信息
     *
     * @param objectName 文件全路径名称
     * @param uploadId   返回的uploadId
     * @param bucketName 桶名称
     * @return Mono<Map < String, Object>>
     */
    public static List<Integer> getChunkByFileMD5(MinioClient client,
String objectName, String uploadId, String bucketName) {
        log.info("通过 <{}-{}-{}> 查询<minio>上传分片数据", objectName,
uploadId, bucketName);
        try {
            // 查询上传后的分片数据
            ListPartsResponse partResult =
```

```
getCustomMinioClient(client).listMultipart(bucketName, null, objectName,
1000, 0, uploadId, null, null);
            return
partResult.result().partList().stream().map(Part::partNumber).collect(Colle
ctors.toList());
        } catch (Exception e) {
            log.error("error message: 查询上传后的分片信息失败、原因:", e);
            return null;
        }
    }

    private static CustomMinioClient getCustomMinioClient(MinioClient
client) {
        CustomMinioClient customMinioClient = new
CustomMinioClient(client);
        return customMinioClient;
    }

    /**
     * 单文件签名上传
     *
     * @param url 文件全路径名称
     * @return /
     */
    public static Map<String, Object> getUploadObjectUrl(String url) {
        try {
            log.info("tip message: 通过 <{}-{}> 开始单文件上传<minio>", url);
            Map<String, Object> resMap = new HashMap();
            List<String> partList = new ArrayList<>();
            log.info("tip message: 单个文件上传、成功");
            partList.add(url);
            resMap.put("uploadId", "SingleFileUpload");
            resMap.put("urlList", partList);
            return resMap;
        } catch (Exception e) {
            log.error("error message: 单个文件上传失败、原因:", e);
            // 返回 文件上传失败
            return null;
        }
    }

    /**
     * 初始化分片上传
     *
     * @param fileUploadInfo
     * @param objectName     文件全路径名称
     * @param chunkNum       分片数量
     * @param contentType    类型，如果类型使用默认流会导致无法预览
```

```
     * @param bucketName     桶名称
     * @return Mono<Map < String, Object>>
     */
    public static Map<String, Object> initMultiPartUpload(MinioClient
minioClient, FileUploadInfo fileUploadInfo, String objectName, int
chunkNum, String contentType, String bucketName) {
        log.info("tip message: 通过 <{}-{}-{}-{}> 开始初始化<分片上传>数据",
objectName, chunkNum, contentType, bucketName);
        Map<String, Object> resMap = new HashMap<>();
        try {
            if (CharSequenceUtil.isBlank(contentType)) {
                contentType = "application/octet-stream";
            }
            HashMultimap<String, String> headers = HashMultimap.create();

            headers.put("Content-Type", contentType);
            CustomMinioClient client = getCustomMinioClient(minioClient);
            //获取uploadId
            String uploadId = null;
            if (StringUtils.isBlank(fileUploadInfo.getUploadId())) {
                uploadId = client.initMultiPartUpload(bucketName, null,
objectName, headers, null);
            } else {
                uploadId = fileUploadInfo.getUploadId();
            }

            resMap.put("uploadId", uploadId);

            fileUploadInfo.setUploadId(uploadId);
            fileUploadInfo.setChunkNum(chunkNum);

            List<String> partList = new ArrayList<>();

            Map<String, String> reqParams = new HashMap<>();
            reqParams.put("uploadId", uploadId);
            for (int i = 1; i <= chunkNum; i++) {
                reqParams.put("partNumber", String.valueOf(i));
                String uploadUrl = client.getPresignedObjectUrl(
                    GetPresignedObjectUrlArgs.builder()
                        .method(Method.PUT)
                        .bucket(bucketName)
                        .object(objectName)
                        .expiry(1, TimeUnit.DAYS)
                        .extraQueryParams(reqParams)
                        .build());
                partList.add(uploadUrl);
            }
            log.info("tip message: 文件初始化<分片上传>、成功");
```

```
            resMap.put("urlList", partList);
            return resMap;
        } catch (Exception e) {
            log.error("error message: 初始化分片上传失败、原因:", e);
            // 返回 文件上传失败
            throw new ServiceException("上传失败");
        }
    }

    /**
     * 分片上传完后合并
     *
     * @param objectName 文件全路径名称
     * @param uploadId   返回的uploadId
     * @param bucketName 桶名称
     * @return boolean
     */
    public static boolean mergeMultipartUpload(MinioClient minioClient
,String objectName, String uploadId, String bucketName) {
        try {
            log.info("tip message: 通过 <{}-{}-{}> 合并<分片上传>数据",
objectName, uploadId, bucketName);
            //目前仅做了最大1000分片
            Part[] parts = new Part[1000];
            CustomMinioClient client = getCustomMinioClient(minioClient);
            // 查询上传后的分片数据
            ListPartsResponse partResult = client.listMultipart("oa-cloud",
null, objectName, 1000, 0, uploadId, null, null);
            int partNumber = 1;
            for (Part part : partResult.result().partList()) {
                parts[partNumber - 1] = new Part(partNumber, part.etag());
                partNumber++;
            }
            // 合并分片
            client.mergeMultipartUpload(bucketName, null,objectName,
uploadId, parts, null, null);

        } catch (Exception e) {
            log.error("error message: 合并失败、原因:", e);
            //TODO删除redis的数据
            return false;
        }
        return true;
    }
}

```

核心功能

核心功能就是CustomMinioClient的初始化的预签名和合并分片

```
package org.springblade.resource.utils;


import com.google.common.collect.Multimap;
import io.minio.CreateMultipartUploadResponse;
import io.minio.ListPartsResponse;
import io.minio.MinioClient;
import io.minio.ObjectWriteResponse;
import io.minio.messages.Part;

public class CustomMinioClient extends MinioClient {

    /**
     * 继承父类
     * @param client
     */
    public CustomMinioClient(MinioClient client) {
        super(client);
    }


    /**
     * 初始化分片上传、获取 uploadId
     *
     * @param bucket           String  存储桶名称
     * @param region           String
     * @param object           String   文件名称
     * @param headers          Multimap<String, String> 请求头
     * @param extraQueryParams Multimap<String, String>
     * @return String
     */
    public String initMultiPartUpload(String bucket, String region, String
object, Multimap<String, String> headers, Multimap<String, String>
extraQueryParams) throws Exception {
        CreateMultipartUploadResponse response =
this.createMultipartUpload(bucket, region, object, headers,
extraQueryParams);
        return response.result().uploadId();
    }
```

```
    /**
     * 合并分片
     *
     * @param bucketName       String   桶名称
     * @param region           String
     * @param objectName       String   文件名称
     * @param uploadId         String   上传的 uploadId
     * @param parts            Part[]   分片集合
     * @param extraHeaders     Multimap<String, String>
     * @param extraQueryParams Multimap<String, String>
     * @return ObjectWriteResponse
     */
    public ObjectWriteResponse mergeMultipartUpload(String
bucketName, String region, String objectName, String uploadId, Part[]
parts, Multimap<String, String> extraHeaders, Multimap<String, String>
extraQueryParams) throws Exception {
        return this.completeMultipartUpload(bucketName, region,
objectName, uploadId, parts, extraHeaders, extraQueryParams);
    }

    /**
     * 查询当前上传后的分片信息
     *
     * @param bucketName       String   桶名称
     * @param region           String
     * @param objectName       String   文件名称
     * @param maxParts         Integer  分片数量
     * @param partNumberMarker Integer  分片起始值
     * @param uploadId         String   上传的 uploadId
     * @param extraHeaders     Multimap<String, String>
     * @param extraQueryParams Multimap<String, String>
     * @return ListPartsResponse
     */
    public ListPartsResponse listMultipart(String bucketName, String
region, String objectName, Integer maxParts, Integer partNumberMarker,
String uploadId, Multimap<String, String> extraHeaders,
Multimap<String, String> extraQueryParams) throws Exception {
        return this.listParts(bucketName, region, objectName, maxParts,
partNumberMarker, uploadId, extraHeaders, extraQueryParams);
    }


}

```

问题解决

配合前端实现了接口之后，发现出现了上传接口的性能问题
由于我们使用的是预签名地址，前端采取了直接调用minio的put接口的方式上传文件，但是minio经常报连接错误

首先先考虑优化minio性能

Minio性能优化：解决“A timeout exceeded while waiting to proceed with the request, please reduce your request”问题

调整Minio配置：检查Minio服务器的配置，确保它已配置为处理高并发请求和高数据量。

* `minio.requests.threads`：增加此参数的值可以增加处理并发请求的线程数。
* `minio.requests.keepalive`：增加此参数的值可以允许服务器保持更多的活动连接。
* `minio.requests.timeout`：增加此参数的值可以增加请求的超时时间。

调整参数后，还可以考虑搭建minio集群

但是发现问题还是会出现上传失败的情况

解决浏览器的瓶颈

假设有10个分片，那我们若是直接发10个请求的话，很容易达到浏览器的瓶颈，所以需要前端对请求进行并发处理。

并发处理：这里使用for循环控制并发的初始并发数，然后在 handler 函数里调用自己，这样就控制了并发

重试: retryArr 数组存储每个切片文件请求的重试次数，做累加。比如[1,0,2],就是第0个文件切片报错1次，第2个报错2次。为保证能与文件做对应，const index = formInfo.index; 我们直接从数据中拿之前定义好的index。 若失败后，将失败的请求重新加入队列即可。

删除脏文件问题

上传失败后，或者中断后产生的脏文件需要处理，可以再补充一个清理文件的接口，合并失败或者长时间未响应的时机去清理文件

解决完以上问题，基本就可以使用了，但是考虑到更多的优化，比如上传文件的进度条，进度条的展示问题，等等，遇见再解决吧

原文链接: https://juejin.cn/post/7371297065739599899