# Minio服务器

<https://www.minio.org.cn/>

阿里云对于对于对象存储服务的定义是：**对象存储服务（Object Storage Service，OSS）是一种海量、安全、低成本、高可靠的云存储服务，适合存放任意类型的文件。**容量和处理能力弹性扩展，多种存储类型供选择，全面优化存储成本。

如果不选择存储上云，那么 Minio 是个不错的选择，它是一个轻量级的服务，作为对象存储的功能基本都具有，且部署简单、支持大容量存储、读写性能优异等特点

# 1. Minio 的基本概念

1. **Object**： **存储到 Minio 的基本对象，如文件、字节流**

1. **Bucket**：**桶（用于存储），用来存储 Object 的逻辑空间，每个 Bucket 之间的数据是相互隔离的**

1. **Drive**：存储数据的磁盘，在 MinIO 启动时，以参数的方式传入

Minio 中所有的对象数据都会存储在 Drive 里

1. **Set** ：即一组 Drive 的集合，分布式部署根据集群规模自动划分一个或多个 Set ，每个 Set 中的 Drive 分布在不同位置。一个对象存储在一个 Set 上

1. 一个对象存储在一个 Set 上

1. 一个集群划分为多个 Set

1. 一个 Set 包含的 Drive 数量是固定的，默认由系统根据集群规模自动计算得出

1. 一个 Set 中的 Drive 尽可能分布在不同的节点上

# 2. 纠删码

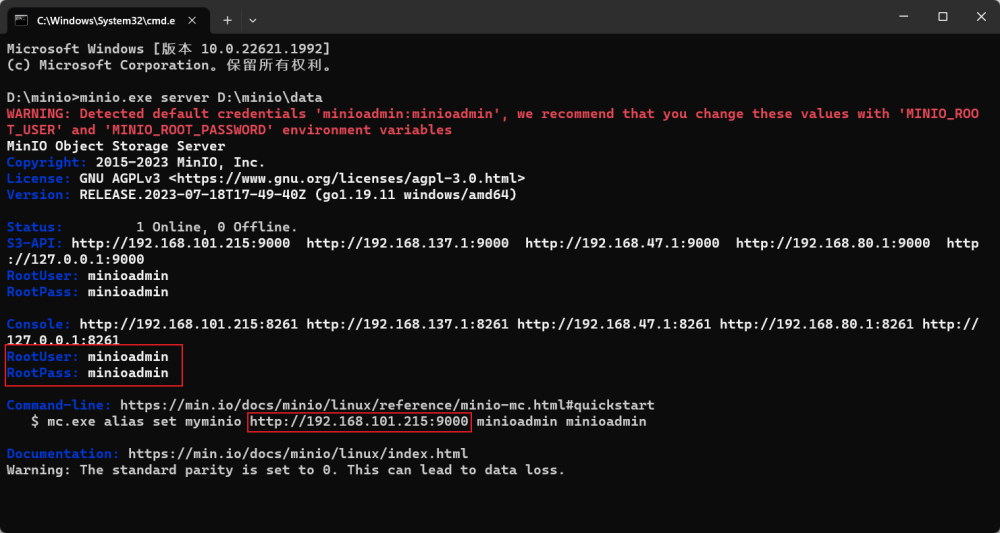
Minio 使用纠删码机制来保证高可靠性，使用 highwayhash 来处理数据损坏； 关于纠删码，简单来说就是可以通过数学计算，把丢失的数据进行还原，它可以将 n 份原始数据，增加 m 份数据，并能通过 n+m 份中的任意n份数据，还原为原始数据。即如果有任意小于等于 m 份的数据失效，仍然能通过剩下的数据还原出来

# 3. Minio 的下载与启动

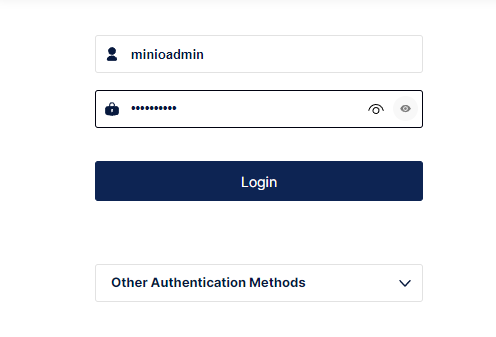
1. 下载文件：https://www.minio.org.cn/

1. 进入到 minio.exe 所在的目录，使用 minio.exe server D:\minio\data

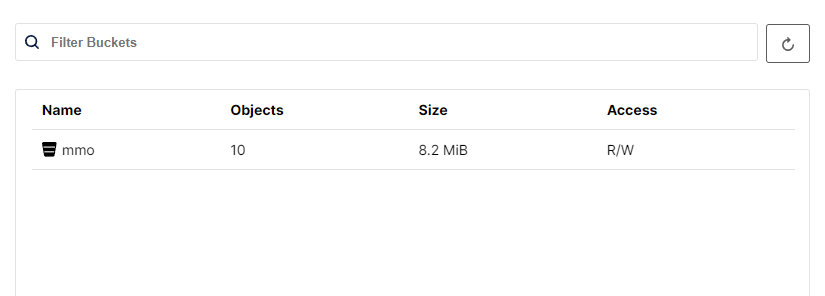
将D:\替换为希望 MinIO 存储数据的驱动器或目录的路径



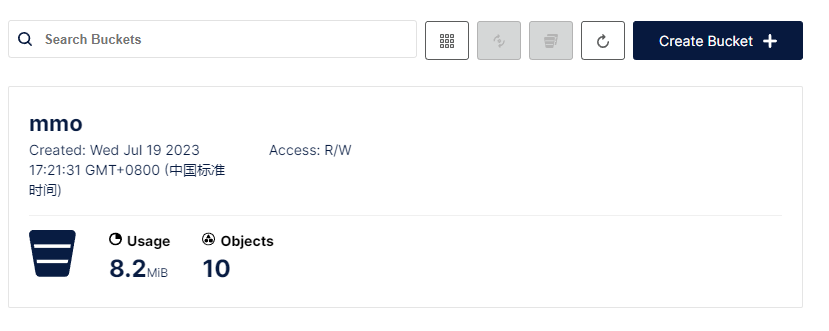
1. 在浏览器输入：http://localhost:9000/ ，进入Minio登录界面



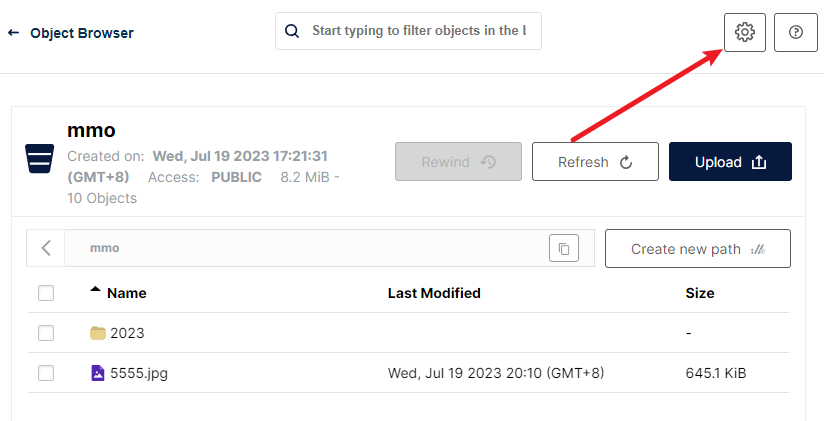
1. 默认的账号和密码都是minioadmin，进入Minio控制台

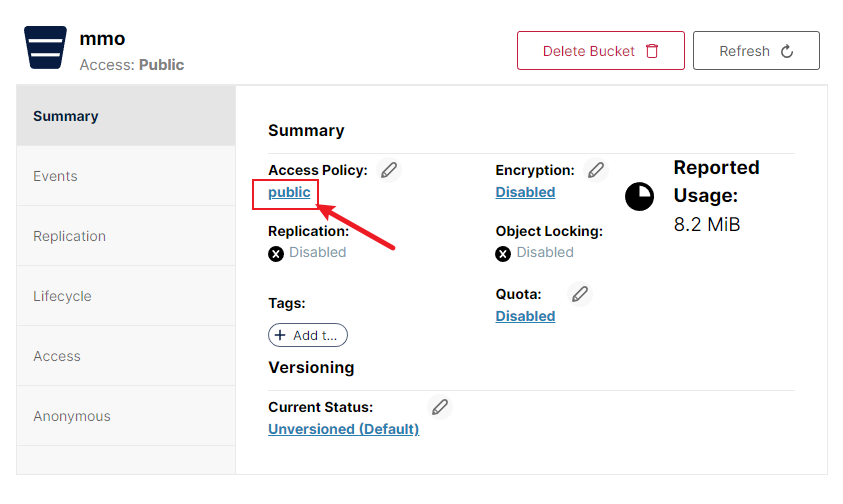


1. 创建 bucket ，Create Bucket



1. 更改权限为 public





# 4. 后台运行流程

<https://blog.csdn.net/qq_36580022/article/details/127898640>

# 5. SpringBoot 整合 minio

1. 导入依赖

<!--minio-->  
<dependency>  
 <groupId>io.minio</groupId>  
 <artifactId>minio</artifactId>  
 <version>8.0.3</version>  
</dependency>

1. yml 配置文件

spring:  
# 配置文件上传大小限制  
 servlet:  
 multipart:  
 max-file-size: 200MB  
 max-request-size: 200MB  
minio:  
 endpoint: http://localhost:9000  
 userName: minioadmin  
 pasword: minioadmin  
 bucketName: mmo  
# minio:  
# endpoint: http://127.0.0.1:9000  
# accessKey: minioadmin  
# secretKey: minioadmin  
# bucketName: mmo

1. 配置类

@Data  
@Component  
@ConfigurationProperties("minio")  
public class MinioProperties {  
 private String endpoint;  
 private String userName;  
 private String password;  
 private String bucketName;  
}

// 注册minio客户端，以便在代码中使用客户端上传文件  
@Data  
@Configuration  
@EnableConfigurationProperties(MinioProperties.class)  
public class MinIOConfig {  
  
 @Resource  
 private MinioProperties minioProperties;  
  
 // 注入minio 客户端  
 @Bean  
 public MinioClient minioClient() {  
  
 return MinioClient.builder()  
 .endpoint(minioProperties.getEndpoint())  
 .credentials(minioProperties.getUserName(), minioProperties.getPassword())  
 .build();  
 }  
  
 @Bean  
 @ConditionalOnBean({MinioClient.class})  
 @ConditionalOnMissingBean(ParallelMinioClient.class)  
 public ParallelMinioClient parallelMinioClient(MinioClient minioClient) {  
 MinioAsyncClient asyncClient = ReflectUtils.getFieldValue(minioClient, "asyncClient");  
 return new ParallelMinioClient(asyncClient);  
 }  
}

/\*\*  
 \* 扩展 MinioClient <很多protected 修饰符的分片方法，MinioClient实例对象无法使用，只能自定义类继承使用>  
 \* minio 大文件分片上传思路：  
 \* 1. 前端访问文件服务，请求上传文件，后端返回签名数据及uploadId  
 \* 2. 前端分片文件，携带签名数据及uploadId并发上传分片数据  
 \* 3. 分片上传完成后，访问合并文件接口，后台负责合并文件。  
 \* @Date 2023-03-14 09:45  
 \*\*/  
public class ParallelMinioClient extends MinioAsyncClient {  
  
 public ParallelMinioClient(MinioAsyncClient client) {  
 super(client);  
 }  
  
 @Override  
 public CompletableFuture<CreateMultipartUploadResponse> createMultipartUploadAsync(String bucketName, String region, String objectName, Multimap<String, String> headers, Multimap<String, String> extraQueryParams) throws InsufficientDataException, InternalException, InvalidKeyException, IOException, NoSuchAlgorithmException, XmlParserException {  
 return super.createMultipartUploadAsync(bucketName, region, objectName, headers, extraQueryParams);  
 }  
  
 @Override  
 public CompletableFuture<UploadPartResponse> uploadPartAsync(String bucketName, String region, String objectName, Object data, long length, String uploadId, int partNumber, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) throws InsufficientDataException, InternalException, InvalidKeyException, IOException, NoSuchAlgorithmException, XmlParserException {  
 return super.uploadPartAsync(bucketName, region, objectName, data, length, uploadId, partNumber, extraHeaders, extraQueryParams);  
 }  
  
 @Override  
 public CompletableFuture<ListPartsResponse> listPartsAsync(String bucketName, String region, String objectName, Integer maxParts, Integer partNumberMarker, String uploadId, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) throws InsufficientDataException, InternalException, InvalidKeyException, IOException, NoSuchAlgorithmException, XmlParserException {  
 return super.listPartsAsync(bucketName, region, objectName, maxParts, partNumberMarker, uploadId, extraHeaders, extraQueryParams);  
 }  
  
 @Override  
 public CompletableFuture<ObjectWriteResponse> completeMultipartUploadAsync(String bucketName, String region, String objectName, String uploadId, Part[] parts, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) throws InsufficientDataException, InternalException, InvalidKeyException, IOException, NoSuchAlgorithmException, XmlParserException {  
 return super.completeMultipartUploadAsync(bucketName, region, objectName, uploadId, parts, extraHeaders, extraQueryParams);  
 }  
  
 @Override  
 public CompletableFuture<AbortMultipartUploadResponse> abortMultipartUploadAsync(String bucketName, String region, String objectName, String uploadId, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) throws InsufficientDataException, InternalException, InvalidKeyException, IOException, NoSuchAlgorithmException, XmlParserException {  
 return super.abortMultipartUploadAsync(bucketName, region, objectName, uploadId, extraHeaders, extraQueryParams);  
 }  
}

1. 实体类

@Data  
public class ObjectItem {  
 private String objectName;  
 private Long size;  
}

@Data  
@Accessors(chain = true)  
public class PartInfo implements Serializable {  
 private int partNumber;  
 private String etag;  
 private String lastModified;  
 private Long size;  
}

@Data  
@Accessors(chain = true)  
public class Task implements Serializable {  
 // 任务id  
 private String id;  
 // 桶名称  
 private String bucketName;  
 // 文件名称  
 private String fileName;  
 // 存储服务器中的文件绝对路径  
 private String remoteFileUrl;  
 // 分片上传的uploadId  
 private String uploadId;  
 // 文件大小（byte）  
 private Long fileSize;  
 // 分片大小（byte）  
 private Long chunkSize;  
 // 分片数量  
 private Long chunkNum;  
 // 上传状态（正在上传：0；已暂停：1；上传成功：2；上传失败：3；终止上传：4）  
 // 说明：暂停、开始都是前端控制，后端只是进行状态记录  
 private String status;  
}

1. 工具类

@Component  
@Slf4j  
@EnableConfigurationProperties(com.ahaxt.base.config.MinioProperties.class)  
public class MinIOUtils extends Base {  
 @Resource  
 private MinioClient minioClient;  
 @Resource  
 private ParallelMinioClient parallelMinioClient;  
 @Resource  
 private SysOssFileDao ossFileDao;  
 @Resource  
 private com.ahaxt.base.config.MinioProperties minioProperties;  
  
 /\*\*  
 \* 判断存储桶是否存在，不存在则创建  
 \*  
 \* @param bucketName 存储桶名称  
 \*/  
 public void existBucket(String bucketName) {  
 try {  
 boolean exists = minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).build());  
 if (!exists) {  
 minioClient.makeBucket(MakeBucketArgs.builder().bucket(bucketName).build());  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 /\*\*  
 \* 查看存储桶是否存在  
 \*  
 \* @param bucketName 存储桶名称  
 \* @return 桶是否存在  
 \*/  
 public boolean bucketExists(String bucketName) {  
 boolean found;  
 try {  
 found = minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).build());  
 } catch (Exception e) {  
 found = false;  
 e.printStackTrace();  
 }  
 return found;  
 }  
  
 /\*\*  
 \* 创建存储桶  
 \*  
 \* @param bucketName 存储桶名称  
 \* @return 是否创建成功  
 \*/  
 public Boolean makeBucket(String bucketName) {  
 try {  
 minioClient.makeBucket(MakeBucketArgs.builder().bucket(bucketName).build());  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 return true;  
 }  
  
 /\*\*  
 \* 删除存储桶  
 \*  
 \* @param bucketName 存储桶名称  
 \* @return 是否删除成功  
 \*/  
 public Boolean removeBucket(String bucketName) {  
 try {  
 minioClient.removeBucket(RemoveBucketArgs.builder().bucket(bucketName).build());  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 return true;  
 }  
  
 /\*\*  
 \* 判断对象是否存在  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param objectName MinIO中存储对象全路径  
 \* @return 对象是否存在  
 \*/  
 public boolean existObject(String bucketName, String objectName) {  
 try {  
 minioClient.statObject(StatObjectArgs.builder().bucket(bucketName).object(objectName).build());  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 return true;  
 }  
  
 /\*\*  
 \* 文件上传  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param file 文件  
 \* @return 桶中位置  
 \*/  
  
  
 /\*\*  
 \* 文件上传  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param file 文件  
 \* @return 桶中位置  
 \*/  
 public SysOssFile upload(String bucketName, String tabName, MultipartFile file, Integer type, Integer relationId) {  
 MultipartFile[] fileArr = {file};  
 List<SysOssFile> fileNames = upload(bucketName, tabName, fileArr, type, relationId);  
 return fileNames.size() == 0 ? null : fileNames.get(0);  
 }  
  
  
 /\*\*  
 \* 上传文件  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param fileList 文件列表  
 \* @return 桶中位置列表  
 \*/  
 public List<SysOssFile> upload(String bucketName, String tabName, List<MultipartFile> fileList, Integer typeId, Integer relationId) {  
 MultipartFile[] fileArr = fileList.toArray(new MultipartFile[0]);  
 return upload(bucketName, tabName, fileArr, typeId, relationId);  
 }  
  
 /\*\*  
 \* description: 上传文件  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param fileArr 文件列表  
 \* @return 桶中位置列表  
 \*/  
 public List<SysOssFile> upload(String bucketName, String tabName, MultipartFile[] fileArr, Integer typeId, Integer relationId) {  
 // 保证桶一定存在  
 existBucket(bucketName);  
 // 执行正常操作  
 List<SysOssFile> bucketFileNames = new ArrayList<>(fileArr.length);  
 for (MultipartFile file : fileArr) {  
 // 获取桶中文件名称  
 // 获取原始文件名称  
 SysOssFile ossFile = new SysOssFile();  
 String originalFileName = file.getOriginalFilename();  
 // 获取当前日期，格式例如：2020/11/04  
 String datePath = new SimpleDateFormat("yyyy/MM-dd/HH").format(new Date());  
 // 文件名称  
 String uuid = UUID.randomUUID().toString().replaceAll("-", "");  
 // 获取文件后缀  
 String type;  
 int index = originalFileName.lastIndexOf(46);  
 if (index != -1) {  
 type = originalFileName.substring(index + 1);  
 } else {  
 type = "unkown";  
 }  
 String bucketFileName = datePath + "/" + uuid + "." + type;  
 ossFile.setBucketName(bucketName);  
 ossFile.setTableName(tabName);  
 ossFile.setFileSize(String.valueOf(file.getSize()));  
 ossFile.setType(typeId);  
 ossFile.setFileName(originalFileName);  
 ossFile.setSuffix(type);  
 ossFile.setOssPath(bucketFileName);  
 ossFile.setRelationId(relationId);  
 ossFile.setUserId(getLoginUserId());  
 // 推送文件到MinIO  
 try (InputStream in = file.getInputStream()) {  
 minioClient.putObject(PutObjectArgs.builder()  
 .bucket(bucketName)  
 .object(bucketFileName)  
 .stream(in, in.available(), -1)  
 .contentType(file.getContentType())  
 .build()  
 );  
 ossFile = ossFileDao.saveAndFlush(ossFile);  
 ossFile.setUrl(getUrl(ossFile));  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 bucketFileNames.add(ossFile);  
 }  
 return bucketFileNames;  
 }  
  
 /\*\*  
 \* 文件下载  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileName 桶中文件名称  
 \* @param originalFileName 原始文件名称  
 \* @param response response对象  
 \*/  
 public void download(String bucketName, String bucketFileName, String originalFileName, HttpServletResponse response) {  
 GetObjectArgs objectArgs = GetObjectArgs.builder().bucket(bucketName).object(bucketFileName).build();  
 try (GetObjectResponse objResponse = minioClient.getObject(objectArgs)) {  
 byte[] buf = new byte[1024];  
 int len;  
 try (FastByteArrayOutputStream os = new FastByteArrayOutputStream()) {  
 while ((len = objResponse.read(buf)) != -1) {  
 os.write(buf, 0, len);  
 }  
 os.flush();  
 byte[] bytes = os.toByteArray();  
 response.setCharacterEncoding("utf-8");  
 //设置强制下载不打开  
 response.setContentType("application/force-download");  
 // 设置附件名称编码  
 originalFileName = new String(originalFileName.getBytes(StandardCharsets.UTF\_8), StandardCharsets.ISO\_8859\_1);  
 // 设置附件名称  
 response.addHeader("Content-Disposition", "attachment;fileName=" + originalFileName);  
 // 写入文件  
 try (ServletOutputStream stream = response.getOutputStream()) {  
 stream.write(bytes);  
 stream.flush();  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 /\*\*  
 \* 获取已上传对象的文件流  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileName 桶中文件名称  
 \* @return 文件流  
 \*/  
 public InputStream getFileStream(String bucketName, String bucketFileName) throws Exception {  
 GetObjectArgs objectArgs = GetObjectArgs.builder().bucket(bucketName).object(bucketFileName).build();  
 return minioClient.getObject(objectArgs);  
 }  
  
 /\*\*  
 \* 批量删除文件对象结果  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileName 桶中文件名称  
 \* @return 删除结果  
 \*/  
 public DeleteError removeObjectsResult(String bucketName, String bucketFileName) {  
 List<DeleteError> results = removeObjectsResult(bucketName, Collections.singletonList(bucketFileName));  
 return results.size() > 0 ? results.get(0) : null;  
 }  
  
 /\*\*  
 \* 批量删除文件对象结果  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileNames 桶中文件名称集合  
 \* @return 删除结果  
 \*/  
 public List<DeleteError> removeObjectsResult(String bucketName, List<String> bucketFileNames) {  
 Iterable<Result<DeleteError>> results = removeObjects(bucketName, bucketFileNames);  
 List<DeleteError> res = new ArrayList<>();  
 for (Result<DeleteError> result : results) {  
 try {  
 res.add(result.get());  
 } catch (Exception e) {  
 e.printStackTrace();  
 log.error("遍历删除结果出现错误：" + e.getMessage());  
 }  
 }  
 return res;  
 }  
  
 /\*\*  
 \* 批量删除文件对象  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileNames 桶中文件名称集合  
 \*/  
 private Iterable<Result<DeleteError>> removeObjects(String bucketName, List<String> bucketFileNames) {  
 List<DeleteObject> dos = bucketFileNames.stream().map(DeleteObject::new).collect(Collectors.toList());  
 return minioClient.removeObjects(RemoveObjectsArgs.builder().bucket(bucketName).objects(dos).build());  
 }  
  
 /\*\*  
 \* 查看文件对象  
 \*  
 \* @param bucketName 存储桶名称  
 \* @return 文件对象集合  
 \*/  
 public List<ObjectItem> listObjects(String bucketName) {  
 Iterable<Result<Item>> results = minioClient.listObjects(  
 ListObjectsArgs.builder().bucket(bucketName).build());  
 List<ObjectItem> objectItems = new ArrayList<>();  
 try {  
 for (Result<Item> result : results) {  
 Item item = result.get();  
 ObjectItem objectItem = new ObjectItem();  
 objectItem.setObjectName(item.objectName());  
 objectItem.setSize(item.size());  
 objectItems.add(objectItem);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 return objectItems;  
 }  
  
 /\*\*  
 \* 获取桶（桶类型：public）中文件访问url  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileName 桶中文件名称  
 \* @return 访问url  
 \*/  
 public String getUploadedObjectUrlForPublicBucket(String bucketName, String bucketFileName) {  
 return bucketName + "/" + bucketFileName;  
 }  
  
 public String getUrl(SysOssFile ossFile) {  
 return minioProperties.getEndpoint() + "/" + ossFile.getBucketName() + "/" + ossFile.getOssPath();  
 }  
  
 /\*\*  
 \* 获取桶（不限制桶类型）中文件访问url  
 \*  
 \* @param bucketName 存储桶名称  
 \* @param bucketFileName 桶中文件名称  
 \* @param expiry 过期时间数量  
 \* @param timeUnit 过期时间单位  
 \* @return 访问url  
 \*/  
 public String getUploadedObjectUrl(String bucketName, String bucketFileName, Integer expiry, TimeUnit timeUnit) {  
 GetPresignedObjectUrlArgs urlArgs = GetPresignedObjectUrlArgs.builder()  
 .method(Method.GET)  
 .bucket(bucketName)  
 .object(bucketFileName)  
 .expiry(expiry, timeUnit)  
 .build();  
 try {  
 return minioClient.getPresignedObjectUrl(urlArgs);  
 } catch (Exception e) {  
 log.error("获取已上传文件的 Url 失败：" + e.getMessage());  
 return "";  
 }  
 }  
  
 /\*\*  
 \* 创建分片上传请求  
 \*  
 \* @param bucketName 桶名称  
 \* @param region 一般填null就行  
 \* @param objectName MinIO中文件全路径  
 \* @param headers 一般只需要设置“Content-Type”  
 \* @return CreateMultipartUploadResponse对象  
 \*\*/  
 public CreateMultipartUploadResponse createMultipartUpload(String bucketName, String region, String objectName, Multimap<String, String> headers, Multimap<String, String> extraQueryParams) {  
 // 保证桶一定存在  
 existBucket(bucketName);  
 // 创建分片上传任务  
 try {  
 return parallelMinioClient.createMultipartUploadAsync(bucketName, region, objectName, headers, extraQueryParams).get();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* 前端通过后端上传分片到MinIO  
 \*  
 \* @param bucketName MinIO桶名称  
 \* @param region 一般填null就行  
 \* @param objectName MinIO中文件全路径  
 \* @param data 分片文件，只能接收RandomAccessFile、InputStream类型的，一般使用InputStream类型  
 \* @param length 文件大小  
 \* @param uploadId 文件上传uploadId  
 \* @param partNumber 分片编号  
 \* @param extraHeaders 一般填null就行  
 \* @param extraQueryParams 一般填null就行  
 \* @return UploadPartResponse对象  
 \*\*/  
 public UploadPartResponse uploadPart(String bucketName, String region, String objectName, Object data, long length, String uploadId, int partNumber, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) {  
 try {  
 return parallelMinioClient.uploadPartAsync(bucketName, region, objectName, data, length, uploadId, partNumber, extraHeaders, extraQueryParams).get();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* 获取分片上传地址，前端直接上传分片到MinIO  
 \*  
 \* @param bucketName MinIO桶名称  
 \* @param ossFilePath MinIO中文件全路径  
 \* @param queryParams 查询参数，一般只需要设置“uploadId”和“partNumber”  
 \* @return 分片上传地址  
 \*\*/  
 public String getPreSignUploadUrl(String bucketName, String ossFilePath, Map<String, String> queryParams) {  
 try {  
 return minioClient.getPresignedObjectUrl(  
 GetPresignedObjectUrlArgs.builder()  
 .method(Method.PUT)  
 .bucket(bucketName)  
 .object(ossFilePath)  
 .expiry(60 \* 60 \* 24)  
 .extraQueryParams(queryParams)  
 .build());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* 获取已上传的所有分片列表，可以为前端和completeMultipartUpload方法服务  
 \*  
 \* @param bucketName MinIO桶名称  
 \* @param region 一般填null就行  
 \* @param ossFilePath MinIO中文件全路径  
 \* @param maxParts 最大分片数，一般填写10000即可  
 \* @param partNumberMarker 直接填0即可  
 \* @param uploadId 文件上传uploadId  
 \* @param extraHeaders 一般填null就行  
 \* @param extraQueryParams 一般填null就行  
 \* @return ListPartsResponse对象  
 \*\*/  
 public ListPartsResponse listParts(String bucketName, String region, String ossFilePath, Integer maxParts, Integer partNumberMarker, String uploadId, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) {  
 try {  
 return parallelMinioClient.listPartsAsync(bucketName, region, ossFilePath, maxParts, partNumberMarker, uploadId, extraHeaders, extraQueryParams).get();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* 合并分片  
 \*  
 \* @param bucketName MinIO桶名称  
 \* @param region 一般填null就行  
 \* @param ossFilePath MinIO中文件全路径  
 \* @param uploadId 文件上传uploadId  
 \* @param parts 分片信息  
 \* @param extraHeaders 一般填null就行  
 \* @param extraQueryParams 一般填null就行  
 \* @return ObjectWriteResponse对象  
 \*\*/  
 public ObjectWriteResponse completeMultipartUpload(String bucketName, String region, String ossFilePath, String uploadId, Part[] parts, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) {  
 try {  
 return parallelMinioClient.completeMultipartUploadAsync(bucketName, region, ossFilePath, uploadId, parts, extraHeaders, extraQueryParams).get();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* 删除MinIO中已有分片  
 \*  
 \* @param bucketName MinIO桶名称  
 \* @param region 一般填null就行  
 \* @param ossFilePath MinIO中文件全路径  
 \* @param uploadId 文件上传uploadId  
 \* @param extraHeaders 一般填null就行  
 \* @param extraQueryParams 一般填null就行  
 \* @return AbortMultipartUploadResponse对象  
 \*\*/  
 public AbortMultipartUploadResponse abortMultipartUpload(String bucketName, String region, String ossFilePath, String uploadId, Multimap<String, String> extraHeaders, Multimap<String, String> extraQueryParams) {  
 try {  
 return parallelMinioClient.abortMultipartUploadAsync(bucketName, region, ossFilePath, uploadId, extraHeaders, extraQueryParams).get();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
}

1. 使用