初识 MinIO

一、 简介

MinIO 是 GlusterFS 创始人之一 Anand Babu Periasamy 发布新的开源项目。基于 Apache License v2.0 开源协议的对象存储项目，采用 Golang 实现，客户端支 Java，Python，Javacript，Golang 语言。

二、运用

MinIO 的应用场景除了可以作为私有云的对象存储服务来使用，也可以作为云对象存储的网关层，无缝对接 Amazon S3 或者 MicroSoft Azure。主要用于存储海量的图片，视频，文档等。非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

三、特点

1、高性能

作为一款高性能存储，在标准硬件条件下，其读写速率分别可以达到 55Gb/s 和 35Gb/s。并且 MinIO 支持一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

2、可扩展

不同 MinIO 集群可以组成联邦，并形成一个全局的命名空间，并且支持跨越多个数据中心。

3、云原生

容器化、基于 K8S 的编排、多租户支持。

4、Amazon S3 兼容

使用 Amazon S3 v2 / v4 API。可以使用 Minio SDK，Minio Client，AWS SDK 和 AWS CLI 访问 Minio 服务器。

5、SDK 支持

GO SDK：https://github.com/minio/minio-go

JavaSDK：https://github.com/minio/minio-java

PythonSDK：https://github.com/minio/minio-py

6、图形化界面

有操作页面。

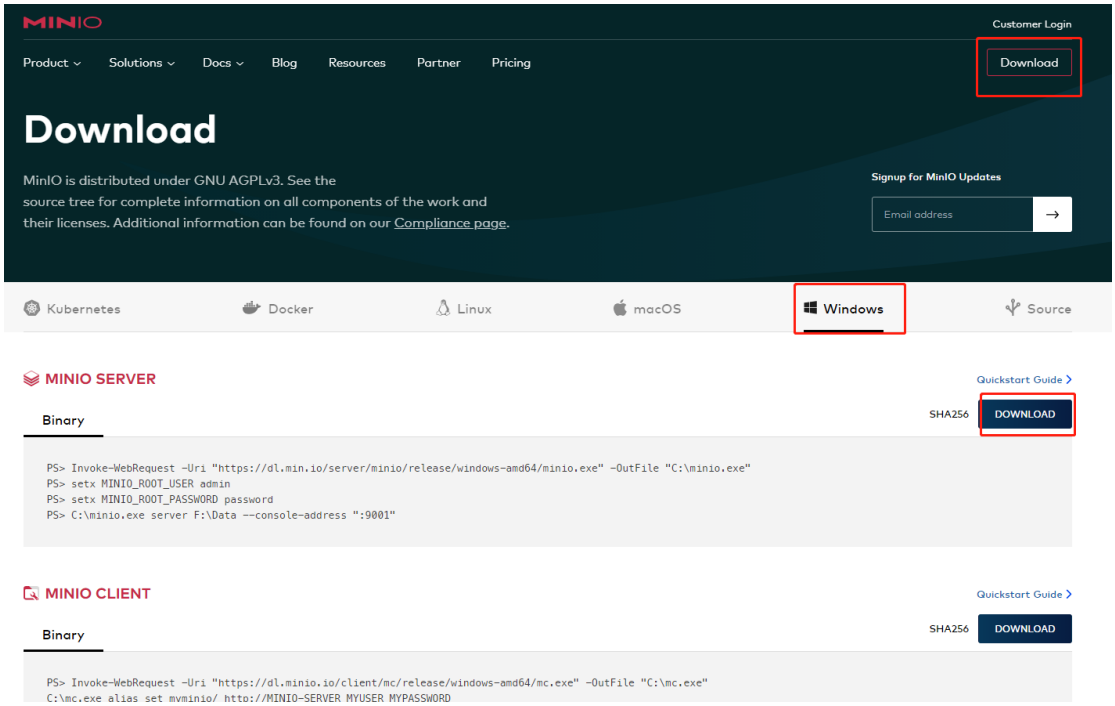
7、支持纠删码

MinIO 使用纠删码、Checksum 来防止硬件错误和静默数据污染。在最高冗余度配置下，即使丢失 1/2 的磁盘也能恢复数据。

四、安装 MinIO

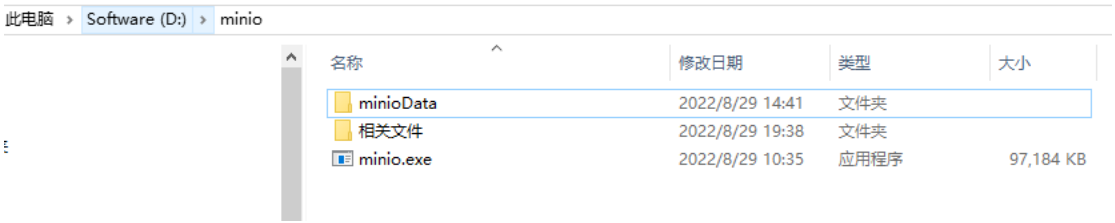
1、下载 MinIO

地址：https://min.io/download#/windows



2、下载如下

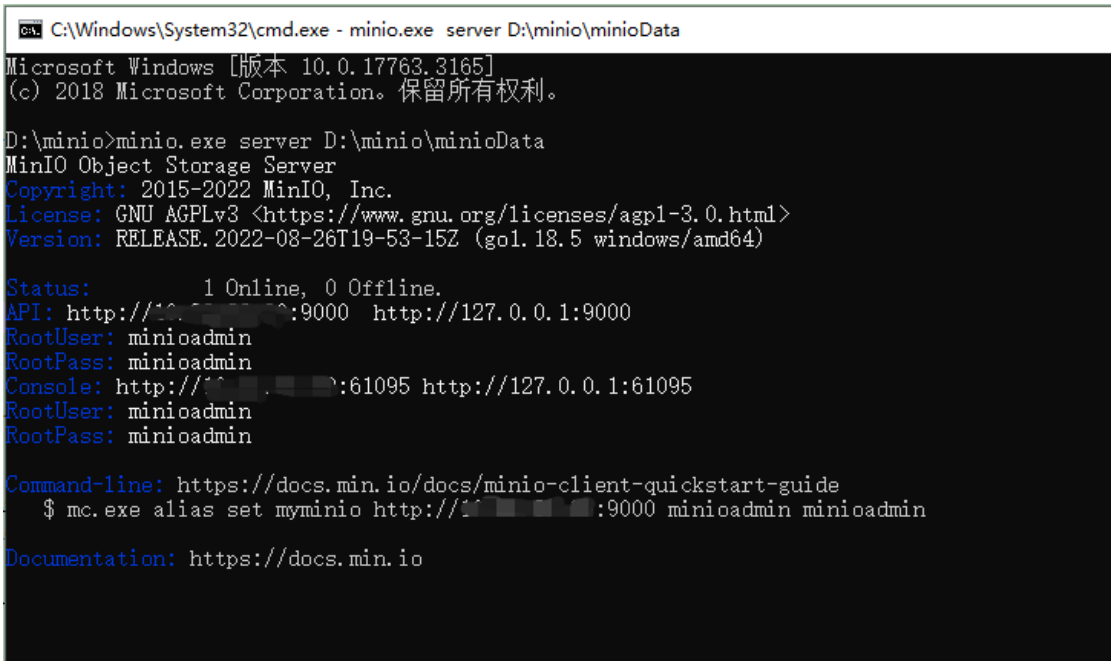
并在当前文件夹下创建 minioData 文件夹，用于存储上传的文件。



3、cmd 进入 dos 命令

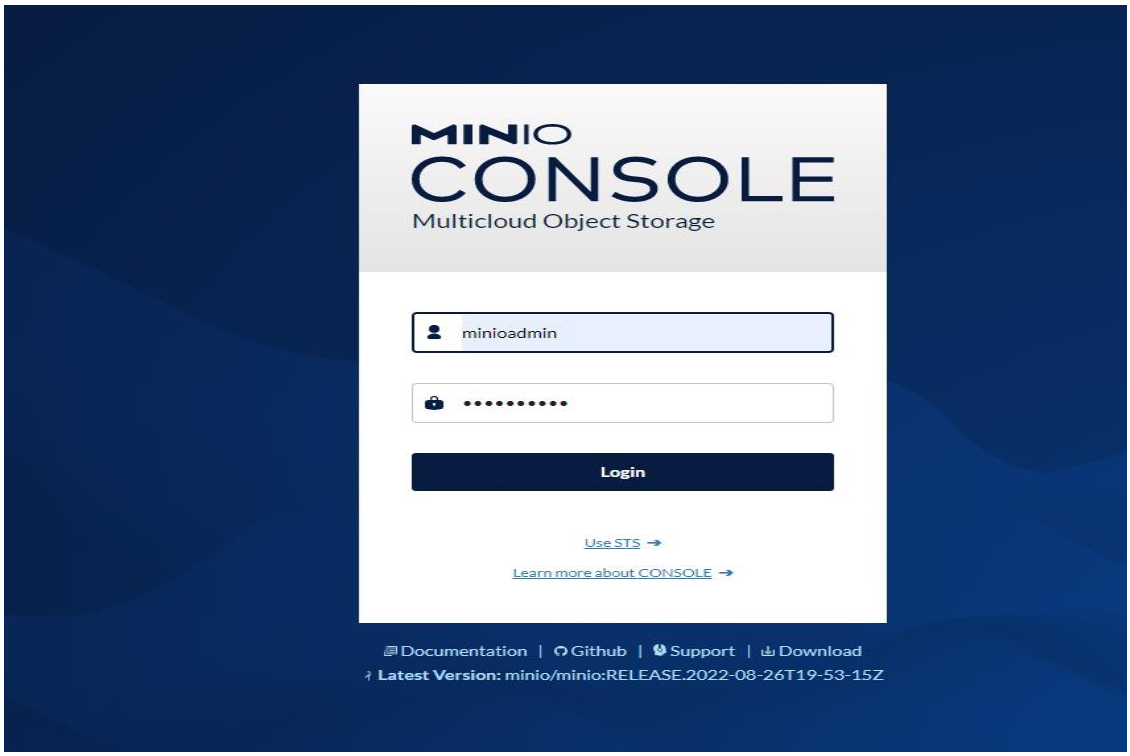
在 minio.exe 文件夹的路径处输入 cmd 进入命令行界面。并执行：minio.exe server

D:\minio\minioData （注意：minioData 取当前电脑的真实路径），启动日志日志如下图所示。



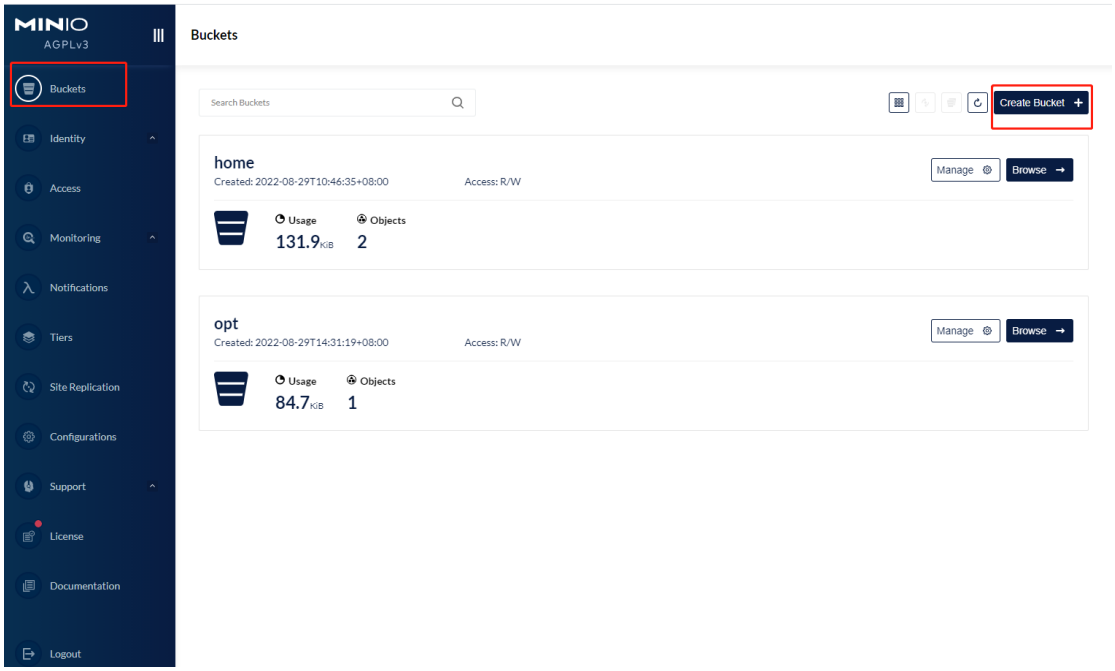
4、minio 启动成功登录

浏览器登陆：http://127.0.0.1:9000，即可进入 minio 界面。默认登录账密：minioadmin || minioadmin



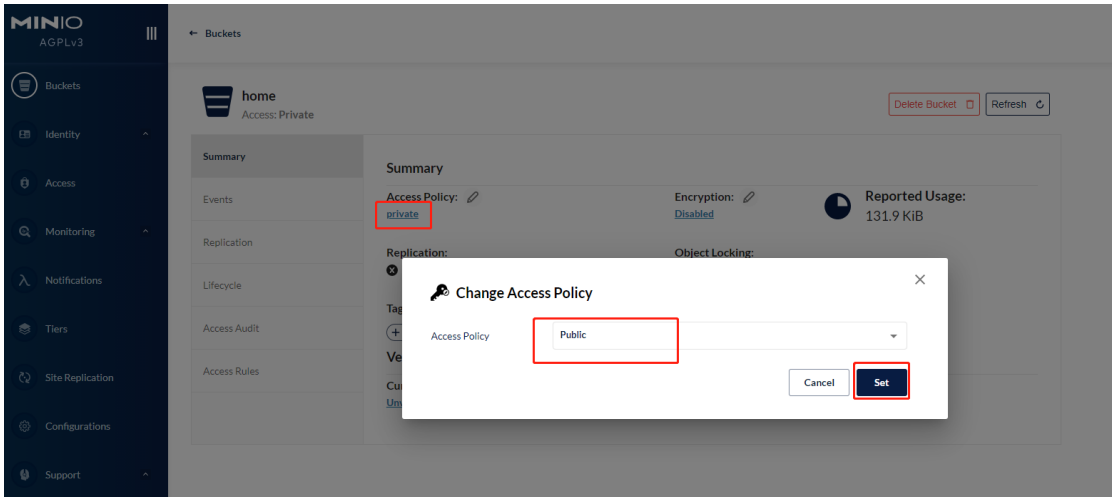
5、创建桶（文件夹）

用于存放上传的文件



6、设置访问权限

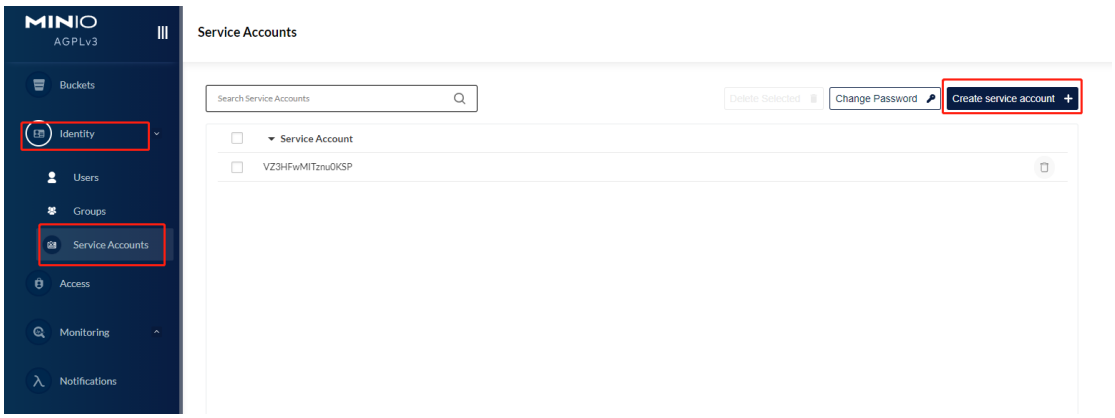
桶的访问权限成 public

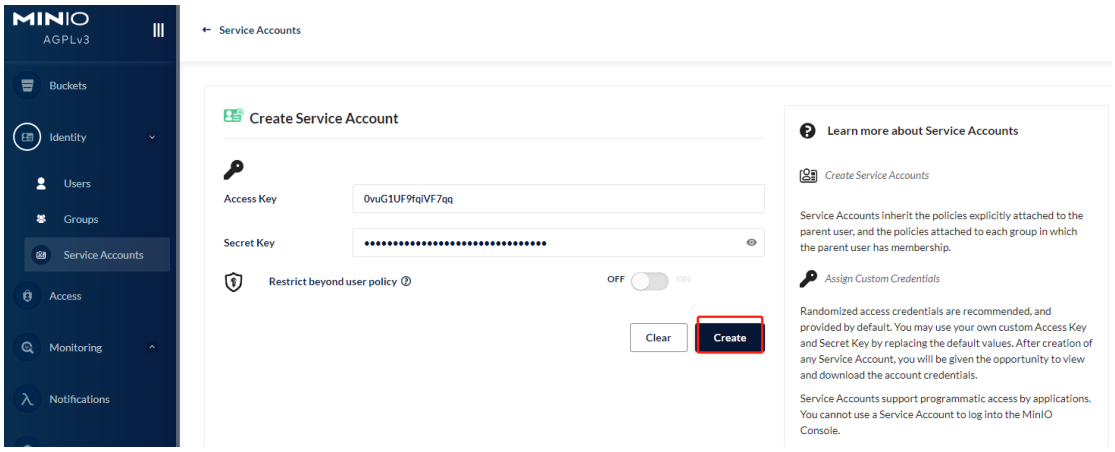


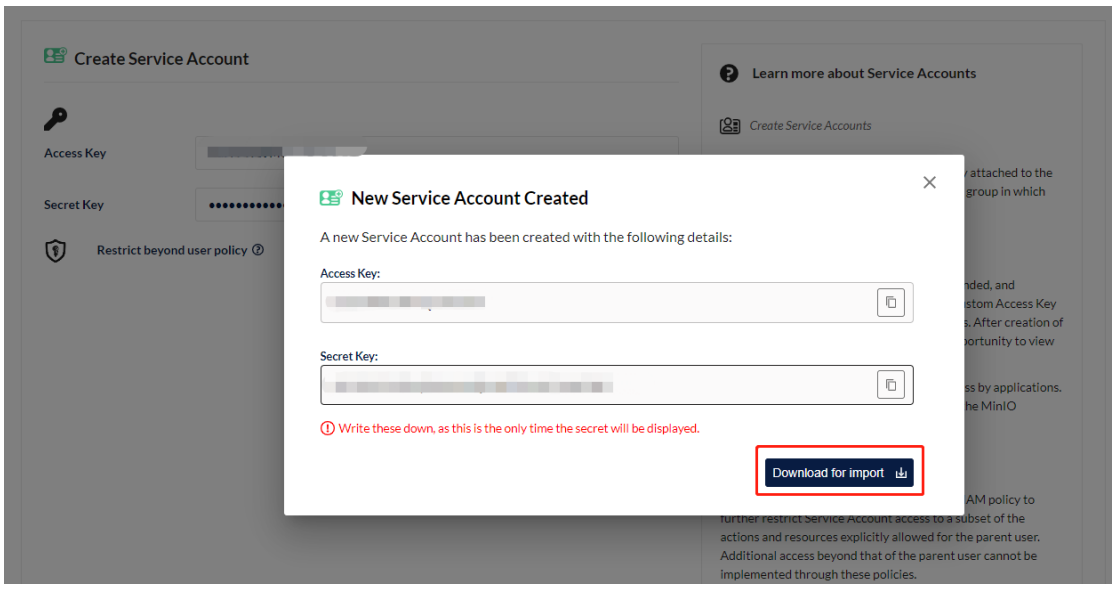
五、SpringBoot 整合 MinIO

1、获取秘钥

accessKey 和 secretKey，保存下来







2、pom.xml 引用

添加 MinIO 依赖

```
<!-- 静态资源服务器，支持文件上传下载-->
<dependency>
    <groupId>io.minio</groupId>
    <artifactId>minio</artifactId>
    <version>8.2.1</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

3、添加配置

秘钥从步骤 1 中的下载文件中获取

```
# 访问的 url
minio.endpoint= http://127.0.0.1
# API 的端口
minio.port=9000
# 秘钥
minio.accessKey=xxx
minio.secretKey=xxxxxxxx
minio.secure=false
# 桶名称
minio.bucket-name=home
# 图片文件的最大大小
minio.image-size=10485760
# 文件的最大大小
minio.file-size=1073741824
```

4、控制层

```
package start.controller;

import org.apache.tomcat.util.http.fileupload.IOUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import start.service.MinIOService;
import start.util.FileTypeUtils;

import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * MinIO 控制层
 */
```

```java
@RequestMapping("/minio")
@RestController
public class MinIOController {

    @Autowired
    private MinIOService minIOService;

    /**
     * 上传文件
     *
     * @param file
     * @param bucketName
     * @return
     */
    @PostMapping("/upload")
    public String uploadFile(MultipartFile file, String bucketName) {
        String fileType = FileTypeUtils.getFileType(file);
        if (fileType != null) {
            return minIOService.putObject(file, bucketName, fileType);
        }
        return "不支持的文件格式。请确认格式，重新上传！！！";
    }

    /**
     * 下载文件
     *
     * @param response
     * @param bucketName
     * @param objectName
     */
    @RequestMapping("/download/{bucketName}/{objectName}")
    public void download(HttpServletResponse response, @PathVariable("bucketName")
String bucketName, @PathVariable("objectName") String objectName) {
        InputStream in = null;
        try {
            in = minIOService.downloadObject(bucketName, objectName);
            response.setHeader("Content-Disposition", "attachment;filename="
                    + URLEncoder.encode(objectName, "UTF-8"));
            response.setCharacterEncoding("UTF-8");
            // 将字节从 InputStream 复制到 OutputStream
            IOUtils.copy(in, response.getOutputStream());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (IOException e) {
```

```
                e.printStackTrace();
            } finally {
                if (in != null) {
                    try {
                        in.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        /**
         * 新增桶(文件夹)
         *
         * @param bucketName
         * @return
         */
        @PostMapping("/addBucket")
        public String addBucket(@RequestParam("bucketName") String bucketName) {
            minIOService.makeBucket(bucketName);
            return "创建成功！！！";
        }

        /**
         * 获取文件列表
         *
         * @param bucketName
         * @return
         */
        @GetMapping("/show/{bucketName}")
        public List<String> show(@PathVariable String bucketName) {
            return minIOService.listObjectNames(bucketName);
        }

        /**
         * 获取桶列表
         *
         * @return
         */
        @GetMapping("/showBucketName")
        public List<String> showBucketName() {
            return minIOService.listBucketName();
        }
```

```
    @GetMapping("/showListObjectNameAndDownloadUrl/{bucketName}")
    public Map<String, String> showListObjectNameAndDownloadUrl(@PathVariable String
bucketName) {
        Map<String, String> map = new HashMap<>();
        List<String> listObjectNames = minIOService.listObjectNames(bucketName);
        String url = "http://localhost:8080/SpringBoot/minio/download/" + bucketName +
"/";
        listObjectNames.forEach(System.out::println);
        for (int i = 0; i < listObjectNames.size(); i++) {
            map.put(listObjectNames.get(i), url + listObjectNames.get(i));
        }
        return map;
    }

    @DeleteMapping("/removeBucket/{bucketName}")
    public String delBucketName(@PathVariable String bucketName) {
        return minIOService.removeBucket(bucketName) == true ? "删除成功" : "删除失败
";
    }

    @DeleteMapping("/removeObject/{bucketName}/{objectName}")
    public String delObject(@PathVariable("bucketName") String bucketName,
@PathVariable("objectName") String objectName) {
        return minIOService.removeObject(bucketName, objectName) == true ? "删除成功
" : "删除失败";
    }

    @DeleteMapping("/removeListObject/{bucketName}")
    public String delListObject(@PathVariable("bucketName") String bucketName,
@RequestBody List<String> objectNameList) {
        return minIOService.removeListObject(bucketName, objectNameList) == true ? "删
除成功" : "删除失败";
    }
}
```

5、接口层

```
package start.service;

import io.minio.messages.Bucket;
```

```java
import org.springframework.web.multipart.MultipartFile;

import java.io.InputStream;
import java.util.List;

/**
 * MinIO 接口类
 */
public interface MinIOService {
    /**
     * 判断 bucket 是否存在
     *
     * @param bucketName
     * @return
     */
    boolean bucketExists(String bucketName);

    /**
     * 创建 bucket
     *
     * @param bucketName
     */
    void makeBucket(String bucketName);

    /**
     * 列出所有存储桶名称
     *
     * @return
     */
    List<String> listBucketName();

    /**
     * 列出所有存储桶 信息
     *
     * @return
     */
    List<Bucket> listBuckets();

    /**
     * 根据桶名删除桶
     *
     * @param bucketName
     */
    boolean removeBucket(String bucketName);
```

```java
    /**
     * 列出存储桶中的所有对象名称
     *
     * @param bucketName
     * @return
     */
    List<String> listObjectNames(String bucketName);

    /**
     * 文件上传
     *
     * @param multipartFile
     * @param bucketName
     */
    String putObject(MultipartFile multipartFile, String bucketName, String fileType);

    /**
     * 文件流下载
     *
     * @param bucketName
     * @param objectName
     * @return
     */
    InputStream downloadObject(String bucketName, String objectName);

    /**
     * 删除文件
     *
     * @param bucketName
     * @param objectName
     */
    boolean removeObject(String bucketName, String objectName);

    /**
     * 批量删除文件
     *
     * @param bucketName
     * @param objectNameList
     * @return
     */
    boolean removeListObject(String bucketName, List<String> objectNameList);

    /**
```

```
        * 获取文件路径
        *
        * @param bucketName
        * @param objectName
        * @return
        */
       String getObjectUrl(String bucketName, String objectName);
}
```

6、实现层

```
package start.service.impl;

import io.minio.messages.Bucket;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import start.constant.MinIOProperties;
import start.service.MinIOService;
import start.util.MinIOUtil;


import java.io.InputStream;
import java.util.List;
import java.util.UUID;

/**
 * MinIO 实现类
 */
@Slf4j
@Service
public class MinIOServiceImpl implements MinIOService {
       @Autowired
       private MinIOUtil minioUtil;

       @Autowired
       private MinIOProperties minioProperties;

       @Override
       public boolean bucketExists(String bucketName) {
              return minioUtil.bucketExists(bucketName);
```

```
    }


    @Override
    public void makeBucket(String bucketName) {
        minioUtil.makeBucket(bucketName);
    }

    @Override
    public List<String> listBucketName() {
        return minioUtil.listBucketNames();
    }

    @Override
    public List<Bucket> listBuckets() {
        return minioUtil.listBuckets();
    }

    @Override
    public boolean removeBucket(String bucketName) {
        return minioUtil.removeBucket(bucketName);
    }


    @Override
    public List<String> listObjectNames(String bucketName) {
        return minioUtil.listObjectNames(bucketName);
    }


    @Override
    public String putObject(MultipartFile file, String bucketName, String fileType) {
        try {
            bucketName = StringUtils.isNotBlank(bucketName) ? bucketName :
minioProperties.getBucketName();
            if (!this.bucketExists(bucketName)) {
                this.makeBucket(bucketName);
            }
            // 文件名
            String fileName = file.getOriginalFilename();
            String objectName = UUID.randomUUID().toString().replaceAll("-", "")
                    + fileName.substring(fileName.lastIndexOf("."));
            minioUtil.putObject(bucketName, file, objectName, fileType);
            return minioProperties.getEndpoint() + ":" + minioProperties.getPort() + "/" +
```

```
bucketName + "/" + objectName;
            } catch (Exception e) {
                log.error("MinIOServiceImpl.putObject() has error", e);
                return "上传失败";
            }
        }

        @Override
        public InputStream downloadObject(String bucketName, String objectName) {
            return minioUtil.getObject(bucketName, objectName);
        }

        @Override
        public boolean removeObject(String bucketName, String objectName) {
            return minioUtil.removeObject(bucketName, objectName);
        }

        @Override
        public boolean removeListObject(String bucketName, List<String> objectNameList) {
            return minioUtil.removeObject(bucketName, objectNameList);
        }

        @Override
        public String getObjectUrl(String bucketName, String objectName) {
            return minioUtil.getObjectUrl(bucketName, objectName);
        }
}
```

7、MinIOUtil 工具类

```
package start.util;

import io.minio.BucketExistsArgs;
import io.minio.GetObjectArgs;
import io.minio.GetPresignedObjectUrlArgs;
import io.minio.ListObjectsArgs;
import io.minio.MakeBucketArgs;
import io.minio.MinioClient;
import io.minio.PutObjectArgs;
import io.minio.RemoveBucketArgs;
import io.minio.RemoveObjectArgs;
import io.minio.RemoveObjectsArgs;
import io.minio.Result;
```

```java
import io.minio.StatObjectArgs;
import io.minio.StatObjectResponse;
import io.minio.http.Method;
import io.minio.messages.Bucket;
import io.minio.messages.DeleteError;
import io.minio.messages.DeleteObject;
import io.minio.messages.Item;
import lombok.SneakyThrows;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;
import start.constant.MinIOProperties;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.TimeUnit;

@Component
public class MinIOUtil {

    private final MinioClient minioClient;

    private final MinIOProperties minioProperties;

    public MinIOUtil(MinioClient minioClient, MinIOProperties minioProperties) {
        this.minioClient = minioClient;
        this.minioProperties = minioProperties;
    }

    /**
     * 检查存储桶是否存在
     *
     * @param bucketName 存储桶名称
     * @return
     */
    @SneakyThrows
    public boolean bucketExists(String bucketName) {
        boolean found =

minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).build());
        if (found) {
            System.out.println(bucketName + " exists");
```

```
        } else {
            System.out.println(bucketName + " does not exist");
        }
        return found;
    }

    /**
     * 创建存储桶
     *
     * @param bucketName 存储桶名称
     */
    @SneakyThrows
    public boolean makeBucket(String bucketName) {
        boolean flag = bucketExists(bucketName);
        if (!flag) {
            minioClient.makeBucket(
                    MakeBucketArgs.builder()
                            .bucket(bucketName)
                            .build());

            return true;
        } else {
            return false;
        }
    }

    /**
     * 列出所有存储桶名称
     *
     * @return
     */
    @SneakyThrows
    public List<String> listBucketNames() {
        List<Bucket> bucketList = listBuckets();
        List<String> bucketListName = new ArrayList<>();
        for (Bucket bucket : bucketList) {
            bucketListName.add(bucket.name());
        }
        return bucketListName;
    }

    /**
     * 列出所有存储桶
     *
```

```
     * @return
     */
    @SneakyThrows
    public List<Bucket> listBuckets() {
        return minioClient.listBuckets();
    }


    /**
     * 删除存储桶
     *
     * @param bucketName 存储桶名称
     * @return
     */
    @SneakyThrows
    public boolean removeBucket(String bucketName) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            Iterable<Result<Item>> myObjects = listObjects(bucketName);
            for (Result<Item> result : myObjects) {
                Item item = result.get();
                // 有对象文件，则删除失败
                if (item.size() > 0) {
                    return false;
                }
            }
            // 删除存储桶，注意，只有存储桶为空时才能删除成功。

minioClient.removeBucket(RemoveBucketArgs.builder().bucket(bucketName).build());
            flag = bucketExists(bucketName);
            if (!flag) {
                return true;
            }
        }
        return false;
    }

    /**
     * 列出存储桶中的所有对象名称
     *
     * @param bucketName 存储桶名称
     * @return
     */
    @SneakyThrows
```

```
    public List<String> listObjectNames(String bucketName) {
        List<String> listObjectNames = new ArrayList<>();
        boolean flag = bucketExists(bucketName);
        if (flag) {
            Iterable<Result<Item>> myObjects = listObjects(bucketName);
            for (Result<Item> result : myObjects) {
                Item item = result.get();
                listObjectNames.add(item.objectName());
            }
        } else {
            listObjectNames.add("存储桶不存在");
        }
        return listObjectNames;
    }


    /**
     * 列出存储桶中的所有对象
     *
     * @param bucketName 存储桶名称
     * @return
     */
    @SneakyThrows
    public Iterable<Result<Item>> listObjects(String bucketName) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            return minioClient.listObjects(
                    ListObjectsArgs.builder().bucket(bucketName).build());
        }
        return null;
    }

    /**
     * 文件上传
     *
     * @param bucketName
     * @param multipartFile
     */
    @SneakyThrows
    public void putObject(String bucketName, MultipartFile multipartFile, String filename,
String fileType) {
        InputStream inputStream = new ByteArrayInputStream(multipartFile.getBytes());
        minioClient.putObject(
                PutObjectArgs.builder().bucket(bucketName).object(filename).stream(
```

```
                                inputStream, -1, minioProperties.getFileSize())
                                .contentType(fileType)
                                .build());
    }


    /**
     * 文件访问路径
     *
     * @param bucketName 存储桶名称
     * @param objectName 存储桶里的对象名称
     * @return
     */
    @SneakyThrows
    public String getObjectUrl(String bucketName, String objectName) {
        boolean flag = bucketExists(bucketName);
        String url = "";
        if (flag) {
            url = minioClient.getPresignedObjectUrl(
                    GetPresignedObjectUrlArgs.builder()
                            .method(Method.GET)
                            .bucket(bucketName)
                            .object(objectName)
                            .expiry(2, TimeUnit.MINUTES)
                            .build());
            System.out.println(url);
        }
        return url;
    }


    /**
     * 删除一个对象
     *
     * @param bucketName 存储桶名称
     * @param objectName 存储桶里的对象名称
     */
    @SneakyThrows
    public boolean removeObject(String bucketName, String objectName) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            minioClient.removeObject(

RemoveObjectArgs.builder().bucket(bucketName).object(objectName).build());
```

```
            return true;
        }
        return false;
    }

    /**
     * 以流的形式获取一个文件对象
     *
     * @param bucketName 存储桶名称
     * @param objectName 存储桶里的对象名称
     * @return
     */
    @SneakyThrows
    public InputStream getObject(String bucketName, String objectName) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            StatObjectResponse statObject = statObject(bucketName, objectName);
            if (statObject != null && statObject.size() > 0) {
                InputStream stream =
                        minioClient.getObject(
                                GetObjectArgs.builder()
                                        .bucket(bucketName)
                                        .object(objectName)
                                        .build());
                return stream;
            }
        }
        return null;
    }

    /**
     * 获取对象的元数据
     *
     * @param bucketName 存储桶名称
     * @param objectName 存储桶里的对象名称
     * @return
     */
    @SneakyThrows
    public StatObjectResponse statObject(String bucketName, String objectName) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            StatObjectResponse stat =
                    minioClient.statObject(
```

```
StatObjectArgs.builder().bucket(bucketName).object(objectName).build());
                return stat;
            }
            return null;
        }

        /**
         * 删除指定桶的多个文件对象,返回删除错误的对象列表，全部删除成功，返回空列表
         *
         * @param bucketName   存储桶名称
         * @param objectNames  含有要删除的多个object 名称的迭代器对象
         * @return
         */
        @SneakyThrows
        public boolean removeObject(String bucketName, List<String> objectNames) {
            boolean flag = bucketExists(bucketName);
            if (flag) {
                List<DeleteObject> objects = new LinkedList<>();
                for (int i = 0; i < objectNames.size(); i++) {
                    objects.add(new DeleteObject(objectNames.get(i)));
                }
                Iterable<Result<DeleteError>> results =
                        minioClient.removeObjects(

RemoveObjectsArgs.builder().bucket(bucketName).objects(objects).build());
                for (Result<DeleteError> result : results) {
                    DeleteError error = result.get();
                    System.out.println(
                            "Error in deleting object " + error.objectName() + "; " +
error.message());
                    return false;
                }
            }
            return true;
        }

        /**
         * 以流的形式获取一个文件对象（断点下载）
         *
         * @param bucketName 存储桶名称
         * @param objectName 存储桶里的对象名称
         * @param offset     起始字节的位置
         * @param length     要读取的长度 (可选，如果无值则代表读到文件结尾)
```

```
     * @return
     */
    @SneakyThrows
    public InputStream getObject(String bucketName, String objectName, long offset, Long
length) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            StatObjectResponse statObject = statObject(bucketName, objectName);
            if (statObject != null && statObject.size() > 0) {
                InputStream stream =
                        minioClient.getObject(
                                GetObjectArgs.builder()
                                        .bucket(bucketName)
                                        .object(objectName)
                                        .offset(offset)
                                        .length(length)
                                        .build());
                return stream;
            }
        }
        return null;
    }


    /**
     * 通过 InputStream 上传对象
     *
     * @param bucketName  存储桶名称
     * @param objectName  存储桶里的对象名称
     * @param inputStream 要上传的流
     * @param contentType 要上传的文件类型 MimeTypeUtils.IMAGE_JPEG_VALUE
     * @return
     */
    @SneakyThrows
    public boolean putObject(String bucketName, String objectName, InputStream
inputStream, String contentType) {
        boolean flag = bucketExists(bucketName);
        if (flag) {
            minioClient.putObject(

PutObjectArgs.builder().bucket(bucketName).object(objectName).stream(
                                    inputStream, -1, minioProperties.getFileSize())
                                    .contentType(contentType)
                                    .build());
```

```
                StatObjectResponse statObject = statObject(bucketName, objectName);
                if (statObject != null && statObject.size() > 0) {
                    return true;
                }
            }
            return false;
        }
}
```

8、FileTypeUtils 工具类

```
package start.util;

import org.springframework.web.multipart.MultipartFile;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLConnection;

/**
 * 文件类型工具类
 * 其中只是定义了部分文件格式，有缺失的自己添加
 */
public class FileTypeUtils {

    private final static String IMAGE_TYPE = "image/";
    private final static String AUDIO_TYPE = "audio/";
    private final static String VIDEO_TYPE = "video/";
    private final static String APPLICATION_TYPE = "application/";
    private final static String TXT_TYPE = "text/";

    public static String getFileType(MultipartFile multipartFile) {
        InputStream inputStream;
        String type;
        try {
            inputStream = multipartFile.getInputStream();
            type = URLConnection.guessContentTypeFromStream(new
BufferedInputStream(inputStream));
            System.out.println(type);
            // 截取'/'之后字符串
```

```
                String prefix = type.substring(0, type.indexOf("/"));
                type = type.substring(prefix.length() + 1);
                if (type.equalsIgnoreCase("JPG") || type.equalsIgnoreCase("JPEG")
                        || type.equalsIgnoreCase("GIF") || type.equalsIgnoreCase("PNG")
                        || type.equalsIgnoreCase("BMP") || type.equalsIgnoreCase("PCX")
                        || type.equalsIgnoreCase("TGA") || type.equalsIgnoreCase("PSD")
                        || type.equalsIgnoreCase("TIFF")) {
                    return IMAGE_TYPE + type;
                }
                if (type.equalsIgnoreCase("mp3") || type.equalsIgnoreCase("OGG")
                        || type.equalsIgnoreCase("WAV") || type.equalsIgnoreCase("REAL")
                        || type.equalsIgnoreCase("APE") ||
type.equalsIgnoreCase("MODULE")
                        || type.equalsIgnoreCase("MIDI") || type.equalsIgnoreCase("VQF")
                        || type.equalsIgnoreCase("CD")) {
                    return AUDIO_TYPE + type;
                }
                if (type.equalsIgnoreCase("mp4") || type.equalsIgnoreCase("avi")
                        || type.equalsIgnoreCase("MPEG-1") || type.equalsIgnoreCase("RM")
                        || type.equalsIgnoreCase("ASF") || type.equalsIgnoreCase("WMV")
                        || type.equalsIgnoreCase("qlv") || type.equalsIgnoreCase("MPEG-2")
                        || type.equalsIgnoreCase("MPEG4") || type.equalsIgnoreCase("mov")
                        || type.equalsIgnoreCase("3gp")) {
                    return VIDEO_TYPE + type;
                }
                if (type.equalsIgnoreCase("doc") || type.equalsIgnoreCase("docx")
                        || type.equalsIgnoreCase("ppt") || type.equalsIgnoreCase("pptx")
                        || type.equalsIgnoreCase("xls") || type.equalsIgnoreCase("xlsx")
                        || type.equalsIgnoreCase("zip") || type.equalsIgnoreCase("jar")) {
                    return APPLICATION_TYPE + type;
                }
                if (type.equalsIgnoreCase("txt")) {
                    return TXT_TYPE + type;
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            return null;
        }
}
```

9、MinIOProperties 配置类

```
package start.constant;

import io.minio.MinioClient;
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;


/**
 * 初始化 MinIO 配置
 */
@Data
@Configuration
@ConfigurationProperties(prefix = "minio")
public class MinIOProperties {

    /**
     * 是一个 URL，域名，IPv4 或者 IPv6 地址")
     */
    private String endpoint;

    /**
     * "TCP/IP 端口号"
     */
    private Integer port;

    /**
     * "accessKey 类似于用户 ID，用于唯一标识你的账户"
     */
    private String accessKey;

    /**
     * "secretKey 是你账户的密码"
     */
    private String secretKey;

    /**
     * "如果是 true，则用的是 https 而不是 http,默认值是 true"
     */
    private boolean secure;
```

```
    /**
     * "默认存储桶"
     */
    private String bucketName;

    /**
     * 图片的最大大小
     */
    private long imageSize;

    /**
     * 其他文件的最大大小
     */
    private long fileSize;


    /**
     * 官网给出的构造方法
     * 此类是客户端进行操作的类
     */
    @Bean
    public MinioClient minioClient() {
        MinioClient minioClient =
                MinioClient.builder()
                        .credentials(accessKey, secretKey)
                        .endpoint(endpoint, port, secure)
                        .build();
        return minioClient;
    }
}
```

10、接口测试

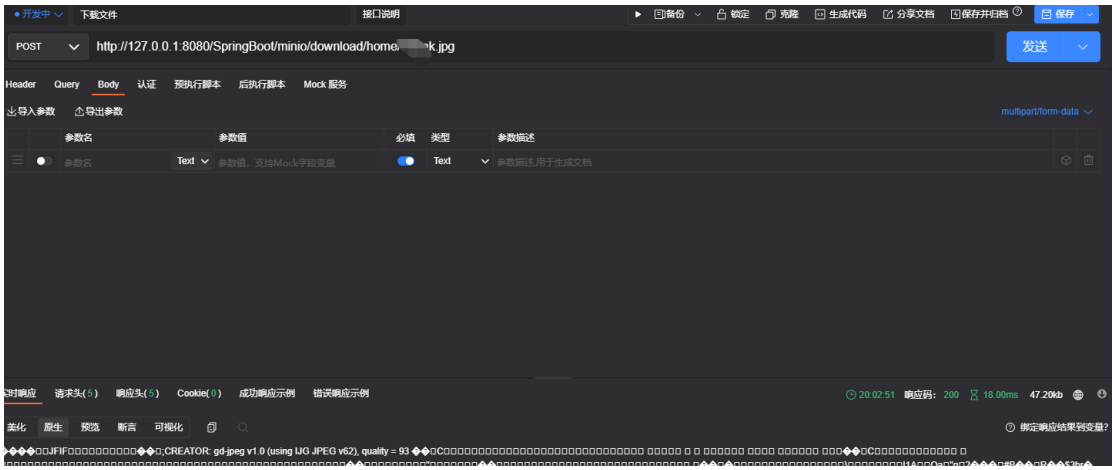
新增桶:


开发中
新增桶
接口说明
备份
锁定
克隆
生成代码
分享文档
保存并归档
保存
POST
http://127.0.0.1:8080/SpringBoot/minio/addBucket?bucketName=opt
发送
Header
Query
Body
认证
预执行脚本
后执行脚本
Mock 服务
美化
简化
提取字段和描述
application/json
1
实时响应
请求头(6)
响应头(5)
Cookie(0)
成功响应示例
错误响应示例
14:40:54
响应码: 200
21.00ms
0.02kb
美化
原生
预览
断言
可视化
1
创建成功！！！

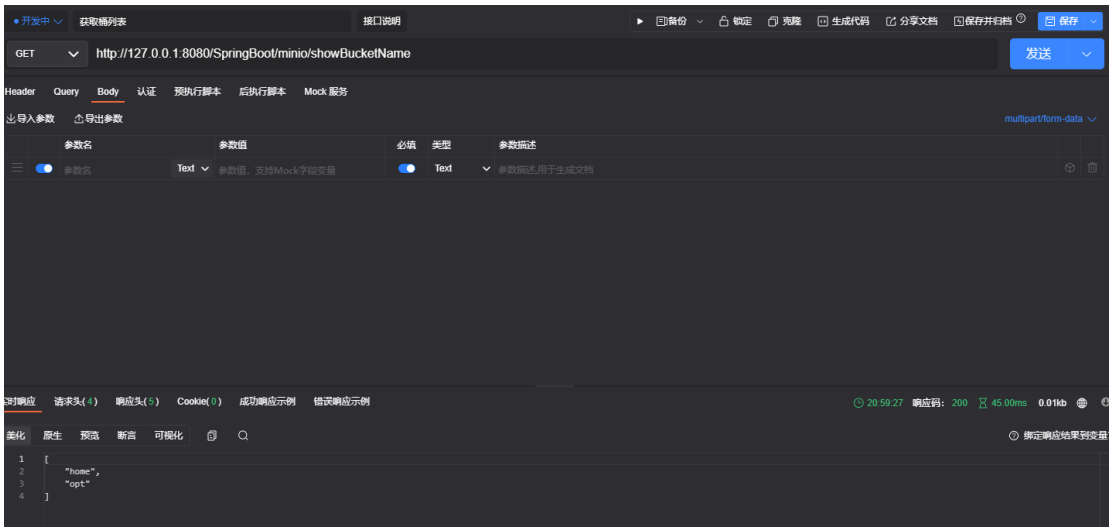
上传文件:


开发中
上传文件
接口说明
备份
锁定
克隆
生成代码
分享文档
保存并归档
保存
POST
http://127.0.0.1:8080/SpringBoot/minio/upload
发送
Header
Query
Body
认证
预执行脚本
后执行脚本
Mock 服务
导入参数
导出参数
multipart/form-data
参数名
参数值
必填
类型
参数描述
bucketName
Text
opt
Text
桶名称
file
File
File
Text
实体文件，类型是File
参数名
Text
参数值，支持Mock字段变量
Text
参数描述用于生成文档
实时响应
请求头(6)
响应头(5)
Cookie(0)
成功响应示例
错误响应示例
15:12:51
响应码: 200
42.00ms
0.06kb
美化
原生
预览
断言
可视化
1
http://
00/opt/21a4c49
1.png

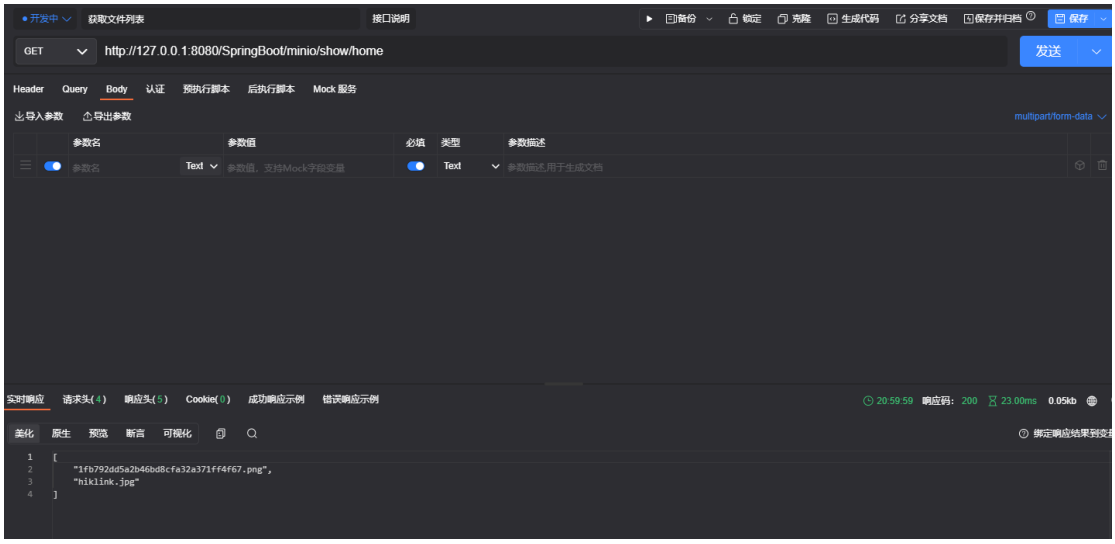
下载文件:



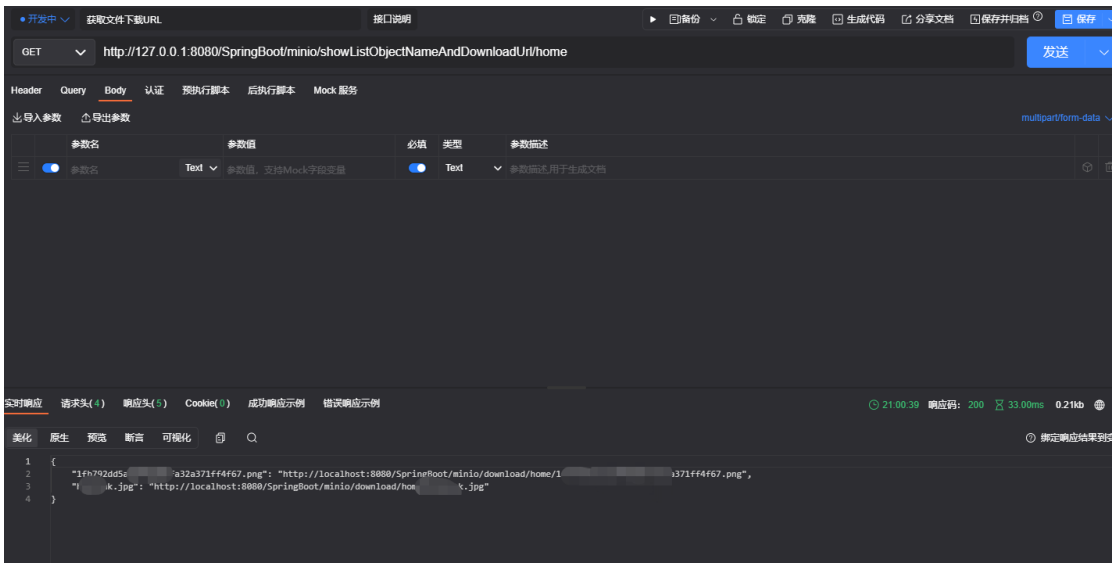
获取桶列表：



获取文件列表：



获取文件下载 URL:



10、以上就是 MinIO 的初步认识