minio

1、minio简介

> MinIO 是一款高性能、分布式的对象存储系统. 它是一款软件产品, 可以100%的运行在标准硬件。即X86等低成本机器也能够很好的运行MinIO

minio官网

在常规的文件上传下载里面，通常我们的流程都是由客户端选择文件，通过流的方式传递到后端，后端写入到硬盘上；写入硬盘有直接写入，或是写入文件服务器ftp之类；

为什么要选择minio?

之前公司项目里面，由于无法使用oss，cos之类的云服务商产品，于是使用的是ftp。上传下载文件都用流传递，来回转换格式，非常麻烦，不仅如此，对于小程序而言，所有的资源都用后台流来发送，这是不合理的，操作起来也比较繁琐，当时就特别希望能使用上oss之类的服务，于是盯上了minio。目的很

简单，一是让文件服务器来承担文件资源的压力，二来是方便小程序引用图片，minio支持https，这样的话，在图片请求上，会方便很多

2、minio安装

2.1、linux安装

服务器安装

使用wget下载包，文档上国内的包无法下载，改用下面的

服务器 Shell 复制代码

```
wget   http://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
./minio server /mnt/data
```

客户端安装

Shell 复制代码

```
wget    http://dl.minio.org.cn/client/mc/release/linux-amd64/mc
chmod +x mc
./mc --help
```

2.2、启动

编辑启动脚本vim minio-start.sh，启动minio，minio本身账户和密码是minioamdin，使用脚本配置启动的用户，并使用--console-address配置控制台ui

启动minio-start.sh Shell 复制代码

```
export MINIO_ROOT_USER=xxx
export MINIO_ROOT_PASSWORD=xxx
./minio server --console-address :49000 /mnt/data > /mnt/data/minio.log 2>&1 &
```

2.3、配置与使用

2.3.1、客户端mc使用

在使用之前，可以使用下面的命令查看minio里面的配置，accesskey和secretkey获取在下方

查看minio服务配置 Shell 复制代码

```
cat ~/.mc/config.json
```

```
[root@MiWiFi-R3600-srv ~]# cat ~/.mc/config.json
{
        "version": "10",
        "aliases": {
                "gcs": {
                        "url": "https://storage.googleapis.com",
                        "accessKey": "YOUR-ACCESS-KEY-HERE",
                        "secretKey": "YOUR-SECRET-KEY-HERE",
                        "api": "S3v2",
                        "path": "dns"
                },
                "local": {
                        "url": "http://192.168.31.171:9000",
                        "accessKey": "vJYjP9cO5EMrsJxp",
                        "secretKey": "BLWD0UyTiBH2Iwla5vRFP1JOx5qpwAgv",
                        "api": "s3v4",
                        "path": "auto"
                },
                "minio": {
                        "url": "http://192.168.31.171:9000",
                        "accessKey": "vJYjP9cO5EMrsJxp",
                        "secretKey": "BLWD0UyTiBH2Iwla5vRFP1JOx5qpwAgv",
                        "api": "s3v4",
                        "path": "auto"
                },
                "play": {
                        "url": "https://play.min.io",
                        "accessKey": "Q3AM3UQ867SPQQA43P2F",
                        "secretKey": "zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG",
                        "api": "S3v4",
                        "path": "auto"
                },
                "s3": {
                        "url": "https://s3.amazonaws.com",
                        "accessKey": "YOUR-ACCESS-KEY-HERE",
                        "secretKey": "YOUR-SECRET-KEY-HERE",
                        "api": "S3v4",
                        "path": "dns"
                }
        }
}[root@MiWiFi-R3600-srv ~]#
```

这里需要注意，官网文档写了api签名是可选的，但实际并不是，如果不加--api s3v4等参数，无法成功执行，且endpoint需要带上端口号，不带端口号，mc后续执行会不成功

配置本地云存储 Shell 复制代码

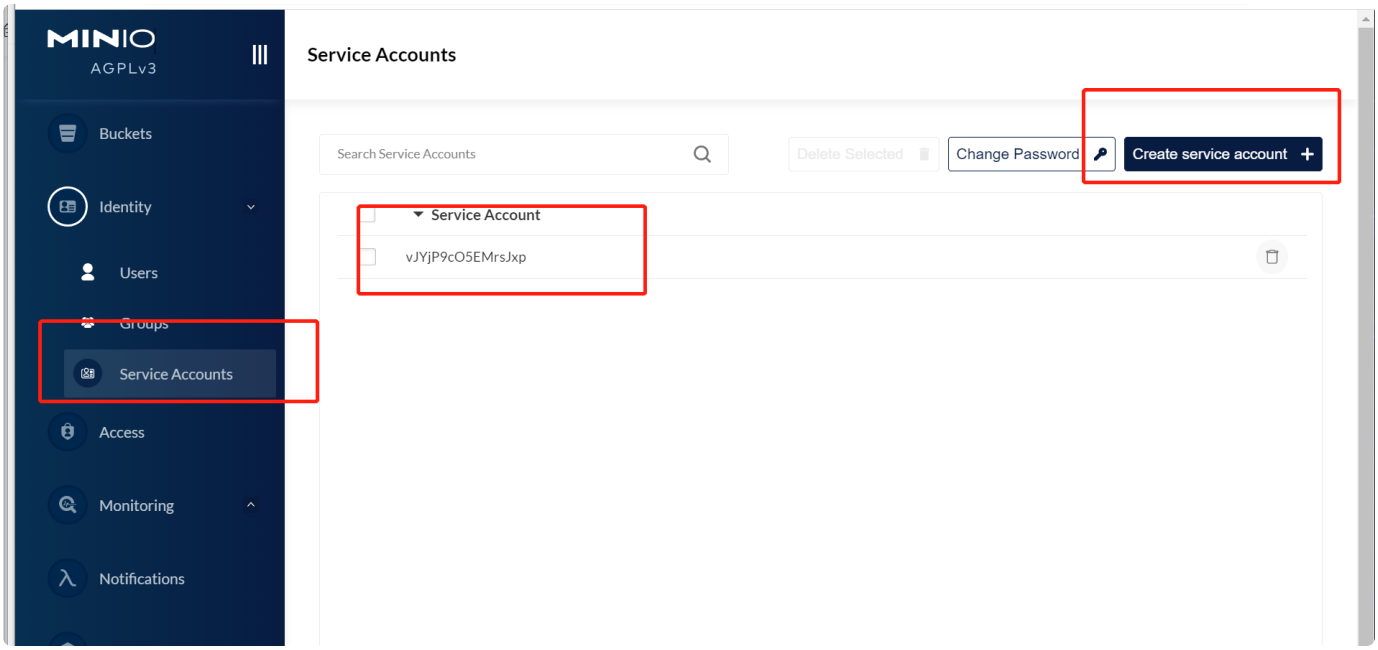
```
mc config host add <ALIAS> <YOUR-S3-ENDPOINT> <YOUR-ACCESS-KEY> <YOUR-SECRET-KEY> [--api API-SIGNATURE]
```

正确示例如下：

Shell 复制代码

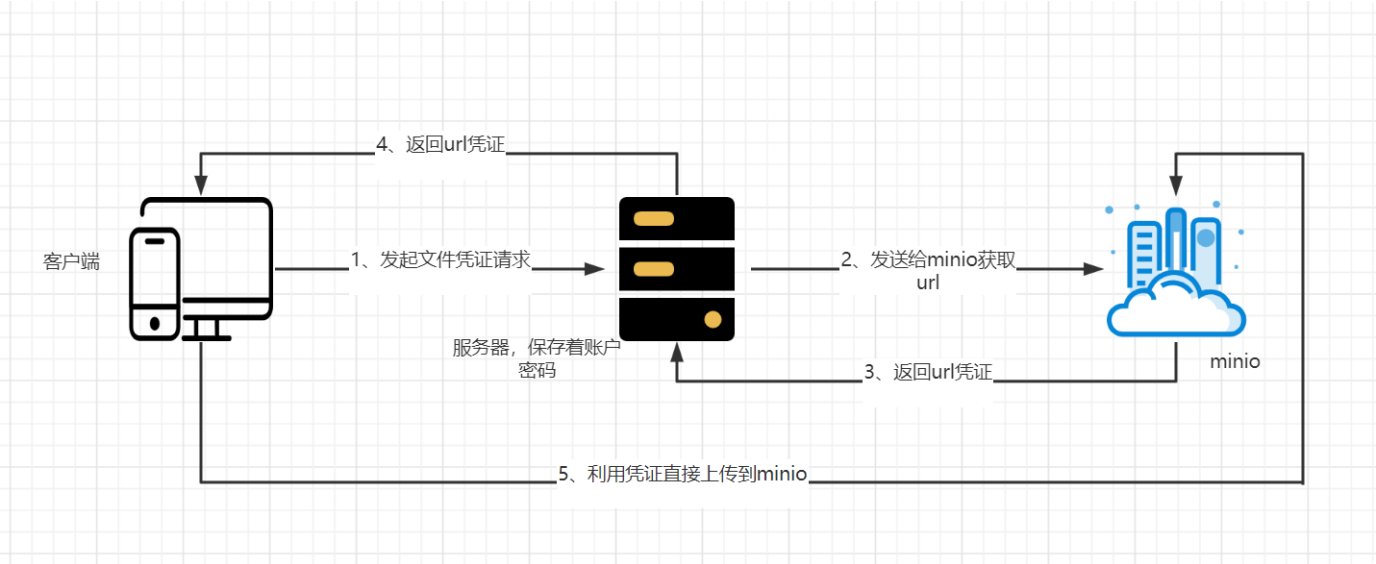
```
./mc config host add minio http://192.168.31.171:9000 vJYjP9cO5EMrsJxp BLWD0UyTiBH2Iwla5vRFP1JOx5qpwAgv --api s3v4
```

此处的accesskey和secretkey可以从控制台生成



3、结合项目使用上传下载

3.1、STS预签名机制

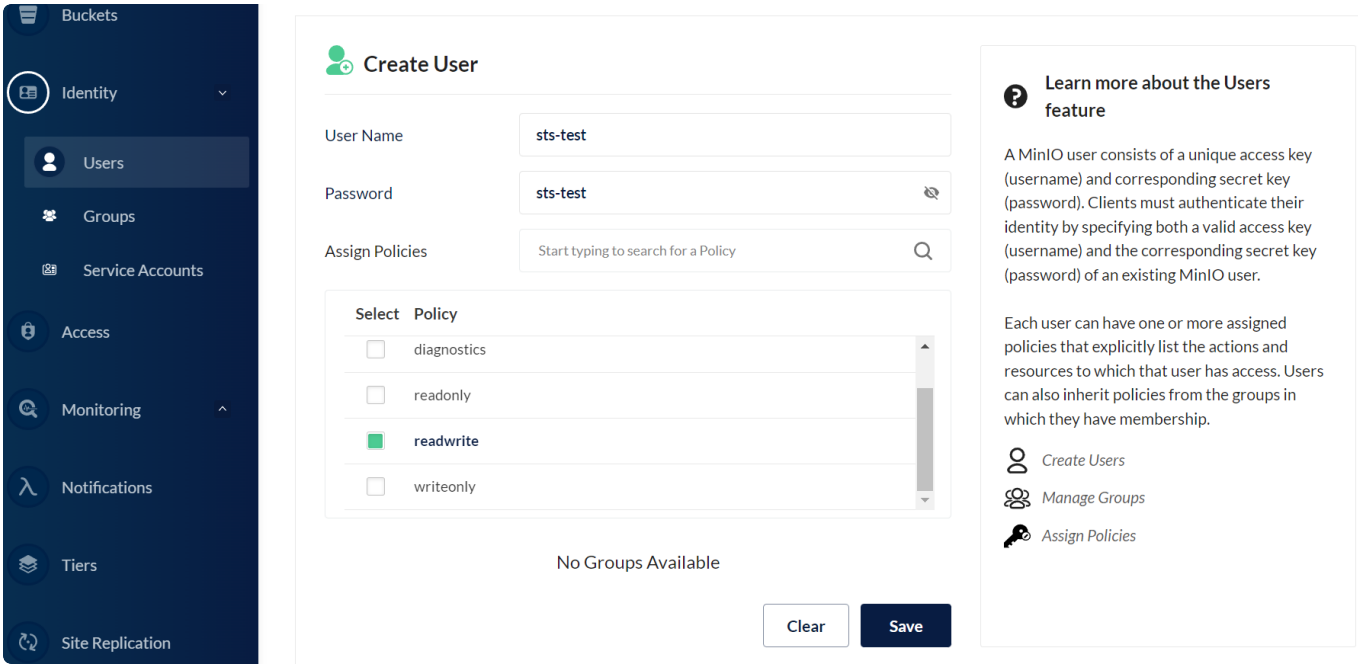


sts流程

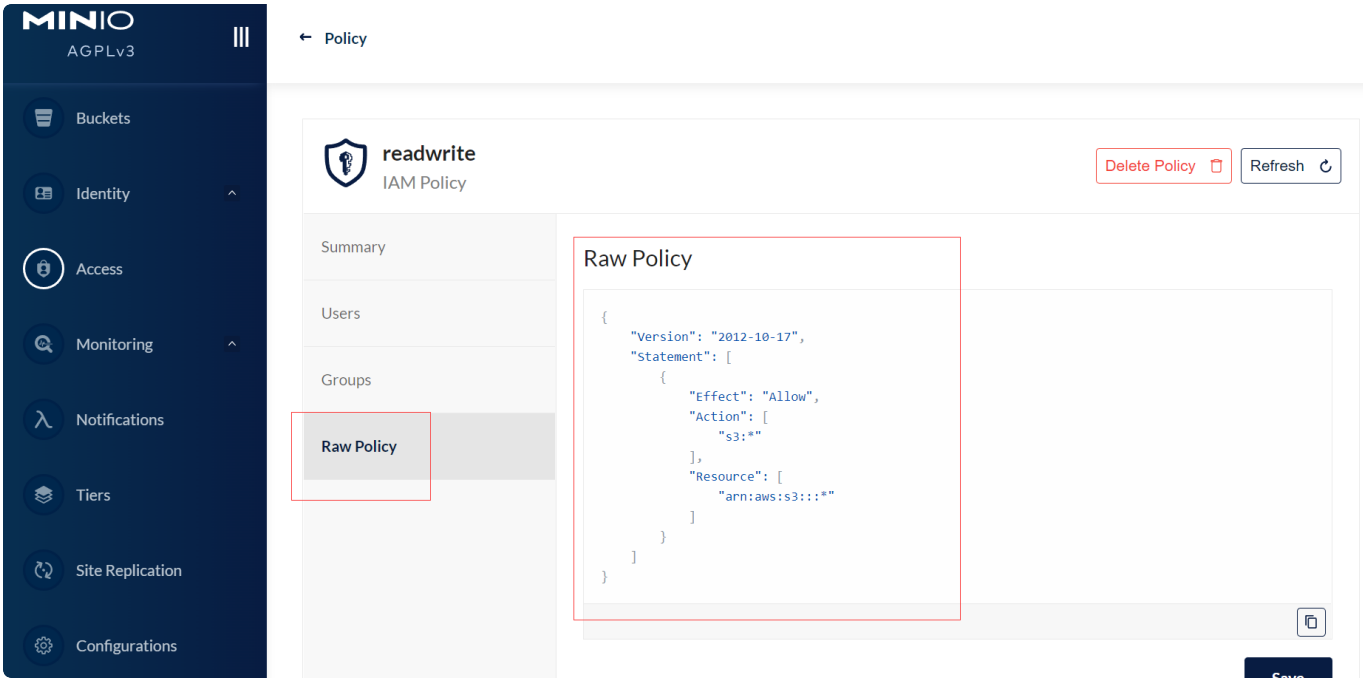
首先，我们来谈谈为什么要使用该机制；在minio上传里面，我们可以选择后端使用sdk上传，或者是前端使用sdk来上传。但是，采用后端上传的话，不可避免的，前端和后端之前会有一次流传递过程，如果文件又大，使用人数又多，很可能导致后台服务出现卡顿，宕机，拖累我们本身的服务；而采用前端上传的话，我们的账户密码会暴露在外面，这样不安全；于是STS机制就来解决这些问题，后端存储我们的账户信息，由前端发起凭证请求，后端将请求发到minio上，minio返回预签名url，后端再将url返回给前端，前端此时利用后端返回的预签名url发起上传动作；在该机制下，前端不会暴露账户，后端也不会承受太大的文件压力，比较均衡

3.1.1、创建用户

登录minio控制台，在identity菜单里面创建用户，我们把账户密码设置成sts-test，选中读写权限来测试，该账户并不需要完整的minio权限



创建用户



创建之后默认的权限配置

3.1.2、代码测试

1、后端代码

在java端，可以直接使用minio-sdk，这里仅写了test代码，正常使用最好自己封装工具类

1. 这里的credentials，可以使用创建的用户账户名和密码，也可以使用生成的service account
2. **这里需要注意的是，如果要用的bucket没有创建，那么一定要先去判断bucket，预签名url不会帮我们创建bucket，会出现异常**
3. minioClient.getPresignedObjectUrl，该api返回的是预处理的url，前端代码可以直接使用url处理，前端必须使用put方法提交，这是aws的规定
4. object参数，是我们要上传到minio的文件名，这个文件名可以根据自己需求写入，可以不和前端选中的文件名相同，最终存在minio的文件以该名称为准

pom依赖 Java 复制代码

```
<!-- https://mvnrepository.com/artifact/io.minio/minio -->
     <!-- 自带的okhttp3版本不符合，需要排除 -->
       <dependency>
           <groupId>io.minio</groupId>
           <artifactId>minio</artifactId>
           <version>8.4.3</version>
           <exclusions>
               <exclusion>
                   <artifactId>okhttp</artifactId>
                   <groupId>com.squareup.okhttp3</groupId>
               </exclusion>
           </exclusions>
       </dependency>
```

java demo Java 复制代码

```java
@Test
void test_sts() throws IOException, InvalidKeyException, InvalidResponseException, InsufficientDataException, NoSuchAlgorithmException, ServerException, InternalException, XmlParserException, ErrorResponseException {
    MinioClient minioClient = MinioClient.builder().credentials("Udo7pnvZzsZfd27u", "ycX7D5CZ5LEwilGzonl5vMQlUABLvkiF")
        .endpoint("http://192.168.31.171:9000").build();
    boolean found =
    minioClient.bucketExists(BucketExistsArgs.builder().bucket("sts-test").build());
    if (!found) {
        // Make a new bucket called 'xxx'.
         minioClient.makeBucket(MakeBucketArgs.builder().bucket("sts-test").build());
    } else {
        System.out.println("Bucket 'sts-test' already exists.");
    }
        //获取sts预url
    String presignedObjectUrl = minioClient.getPresignedObjectUrl(GetPresignedObjectUrlArgs.builder()
                    .method(Method.PUT).object("MySQL环境搭建.pdf").bucket("sts-test").expiry(1, TimeUnit.DAYS).build());
    System.out.println(presignedObjectUrl);
    }
```

预签名方法源码 Java 复制代码

```java
  /**
   * Gets presigned URL of an object for HTTP method, expiry time and custom request parameters.
   *
   * <pre>Example:{@code
   * // Get presigned URL string to delete 'my-objectname' in 'my-bucketname' and its life time
   * // is one day.
   * String url =
   *    minioClient.getPresignedObjectUrl(
   *        GetPresignedObjectUrlArgs.builder()
   *            .method(Method.DELETE)
   *            .bucket("my-bucketname")
   *            .object("my-objectname")
   *            .expiry(24 * 60 * 60)
   *            .build());
   * System.out.println(url);
   *
   * // Get presigned URL string to upload 'my-objectname' in 'my-bucketname'
   * // with response-content-type as application/json and life time as one day.
   * Map<String, String> reqParams = new HashMap<String, String>();
   * reqParams.put("response-content-type", "application/json");
   *
   * String url =
   *    minioClient.getPresignedObjectUrl(
   *        GetPresignedObjectUrlArgs.builder()
   *            .method(Method.PUT)
   *            .bucket("my-bucketname")
   *            .object("my-objectname")
   *            .expiry(1, TimeUnit.DAYS)
   *            .extraQueryParams(reqParams)
   *            .build());
   * System.out.println(url);
   *
   * // Get presigned URL string to download 'my-objectname' in 'my-bucketname' and its life time
   * // is 2 hours.
   * String url =
   *    minioClient.getPresignedObjectUrl(
   *        GetPresignedObjectUrlArgs.builder()
   *            .method(Method.GET)
   *            .bucket("my-bucketname")
   *            .object("my-objectname")
```

```
     *            .expiry(2, TimeUnit.HOURS)
     *            .build());
     * System.out.println(url);
     * }</pre>
     *
     * @param args {@link GetPresignedObjectUrlArgs} object.
     * @return String - URL string.
     * @throws ErrorResponseException thrown to indicate S3 service returned an error response.
     * @throws InsufficientDataException thrown to indicate not enough data available in InputStream.
     * @throws InternalException thrown to indicate internal library error.
     * @throws InvalidKeyException thrown to indicate missing of HMAC SHA-256 library.
     * @throws InvalidResponseException thrown to indicate S3 service returned invalid or no error
     *     response.
     * @throws IOException thrown to indicate I/O error on S3 operation.
     * @throws NoSuchAlgorithmException thrown to indicate missing of MD5 or SHA-256 digest library.
     * @throws XmlParserException thrown to indicate XML parsing error.
     * @throws ServerException
     */
    public String getPresignedObjectUrl(GetPresignedObjectUrlArgs args)
        throws ErrorResponseException, InsufficientDataException, InternalException,
            InvalidKeyException, InvalidResponseException, IOException, NoSuchAlgorithmException,
            XmlParserException, ServerException {
      return asyncClient.getPresignedObjectUrl(args);
    }
```

2、前端代码

要使用minio的sts，我们需要引入aws的sdk，minio使用的接口协议是aws的s3API，是全世界都认可的标准

这里只是简单示范了下关键代码

1. url，来自后端返回的预签名url，在上传时，应该前端先请求后端的获取签名方法，得到url后然后再进行上传
2. 文件不需要转成formdata，直接把文件和url放进去即可

简单的demo JavaScript 复制代码

```
<input type="file" @change="handleFileInputChange" />
<button @click="submitUpload">开始上传</button>

 handleFileInputChange(event) {
    this.file = event.target.files[0];
 },
//这里的url其实是后端返回的
submitUpload(){
  let url = 'http://192.168.31.171:9000/sts-test/MySQL%E7%8E%AF%E5%A2%83%E6%90%AD%E5%BB%BA.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=Udo7pnvZzsZfd27u%2F20220903%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220903T083235Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signature=76a16b7ed9695565d2ad92b9a6c359b6d2f1a8d91345f376ff38249997548445';
  axios.put(url,this.file)
    .then(res=>{
    console.log(res)
  })
},
```

3.2、直传

直传分为后端直传和前端直传，使用方法很简单，官网文档有例子，这里只展示后端部分，前端同理

直传方式在预签名STS上已经说过了，这里不多展开

java直传 Java 复制代码

```java
 @Test
    void contextLoads() throws NoSuchAlgorithmException, InvalidKeyException, IOException {
        try {
            // Create a minioClient with the MinIO server playground, its access key and secret key.
            MinioClient minioClient =
                    MinioClient.builder()
                            .endpoint("http://192.168.31.171:9000")
                            .credentials("xxx", "xxxx")
                            .build();

            // Make 'asiatrip' bucket if not exist.
            boolean found =
                    minioClient.bucketExists(BucketExistsArgs.builder().bucket("gulipdf").build());
            if (!found) {
                // Make a new bucket called 'asiatrip'.
                minioClient.makeBucket(MakeBucketArgs.builder().bucket("gulipdf").build());
            } else {
                System.out.println("Bucket 'gulipdf' already exists.");
            }


            // Upload '/home/user/Photos/asiaphotos.zip' as object name 'asiaphotos-2015.zip' to bucket
            // 'asiatrip'.
            minioClient.uploadObject(
                    UploadObjectArgs.builder()
                            .bucket("gulipdf")
                            .object("写在最后.pdf")
                            .filename("G:\\第00章_写在最后.pdf")
                            .build());
            System.out.println(
                    "'/home/user/Photos/asiaphotos.zip' is successfully uploaded as "
                            + "object 'asiaphotos-2015.zip' to bucket 'asiatrip'.");
        } catch (MinioException e) {
            System.out.println("Error occurred: " + e);
            System.out.println("HTTP trace: " + e.httpTrace());
        }

    }
```