

MinIO

简介

[基本概念](#)

[特点](#)

[安装MinIO](#)

[快速入门](#)

[封装MinIO Starter](#)

简介

MinIO 兼容亚马逊 S3 云存储服务接口，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

基本概念

`bucket`：类比于目录

`object`：类比于文件

`keys`：类比文件名

特点

数据保护

- 使用Minio Erasure Code（纠删码）来防止硬件故障，可以进行数据恢复

高性能

- 在标准硬件条件下它能达到 55GB/s 的读、35GB/s 的写速率

可扩容

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

文件变化主动通知存储桶

- 如果发生改变,比如上传对象和删除对象，可以使用存储桶事件通知机制进行监控
- 并通过以下方式发布出去：
 - AMQP 、 Elasticsearch 、 Redis 、 MySQL 、 Kafkas 等。

安装MinIO

1. 拉取 minio/minio 镜像

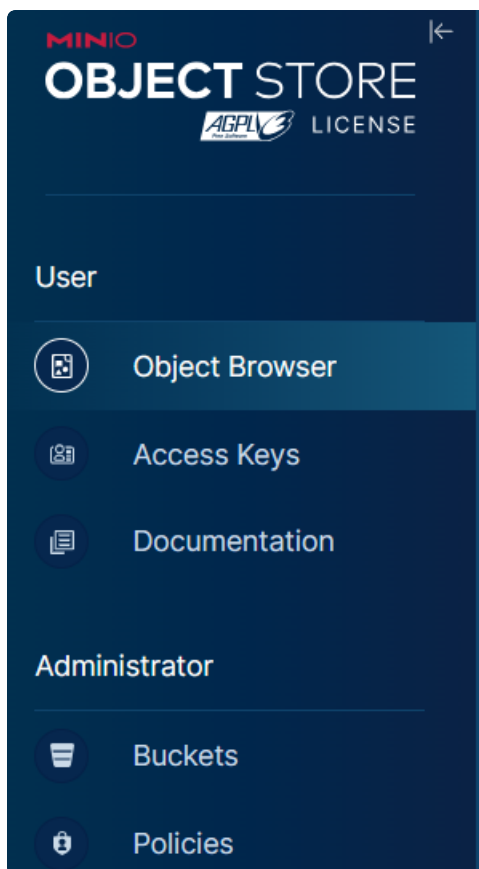
```
1 docker pull minio/minio
```

2. 创建 data , config 目录

```
[root@project minio]# ll
总用量 8
drwxr-xr-x. 3 root root 4096 2月  5 11:45 config
drwxr-xr-x. 2 root root 4096 2月  5 11:41 data
[root@project minio]#
```

3. 允许容器

```
1 docker run \  
2   -p 9000:9000 \  
3   -p 9090:9090 \  
4   --name minio \  
5   -d --restart=always \  
6   -e "MINIO_ACCESS_KEY=minio@123" \  
7   -e "MINIO_SECRET_KEY=minio@123" \  
8   -v /project/leadnews/minio/data:/data \  
9   -v /project/leadnews/minio/config:/root/.minio \  
10  minio/minio server \  
11  /data --console-address ":9090" --address ":9000"
```



Object Browser



Buckets

MinIO uses buckets to organize objects. objects.

To get started, [Create a Bucket](#).

- 创建一个桶(目录)

Q Search Buckets

leadnews

Created: Mon Feb 05 2024 12:13:56 GMT+0800 (中国标准时间)

Access: R/W



Usage

0.0_B

Objects

0

快速入门

1. 导入依赖

▼ pom.xml

XML

```
1 <dependencies>
2   <!-- minio -->
3   <dependency>
4     <groupId>io.minio</groupId>
5     <artifactId>minio</artifactId>
6     <version>8.5.7</version>
7   </dependency>
8   <dependency>
9     <groupId>org.springframework.boot</groupId>
10    <artifactId>spring-boot-starter-web</artifactId>
11  </dependency>
12  <dependency>
13    <groupId>org.springframework.boot</groupId>
14    <artifactId>spring-boot-starter-test</artifactId>
15  </dependency>
16 </dependencies>
```

2. 创建启动类


```
1  /**
2   * @author 发着呆看星
3   * @create 2024/2/5
4   */
5   @SpringBootApplication
6   public class MinIoApplication {
7       public static void main(String[] args) {
8           SpringApplication.run(MinIoApplication.class, args);
9       }
10  }
```

3. 创建测试类，上传文件

```
1  @Test
2  public void uploadTest(){
3      FileInputStream fileInputStream = null;
4      try {
5          // 创建文件输入流，获取要上传的文件
6          fileInputStream = new FileInputStream("D://list.html");
7
8          // 创建 minio连接客户端
9          MinioClient client = MinioClient.builder()
10              .credentials("minio@123", "minio@123")
11              .endpoint("http://192.168.16.166:9000")
12              .build();
13
14          // 上传
15          PutObjectArgs putObjectArgs = PutObjectArgs.builder()
16              .object("list.html")
17              .contentType("text/html")
18              .bucket("leadnews")
19              .stream(fileInputStream, fileInputStream.available(), -1)
20              .build();
21          client.putObject(putObjectArgs);
22
23          System.out.println("上传成功");
24      } catch (Exception e) {
25          throw new RuntimeException(e);
26      }
27  }
```

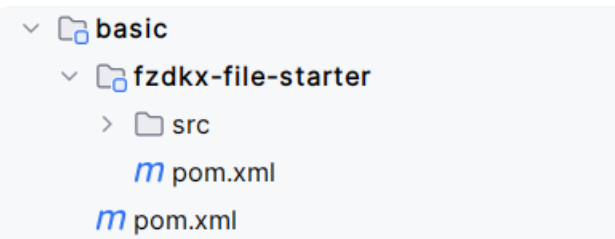
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only

上传成功

Name	Objects	Size
 leadnews	1	1.9 KiB

封装MinIO Starter

1. 创建模块



2. 引入依赖

```
1  <!-- 依赖版本在父模块已经统一配置 -->
2  <dependencies>
3      <!--spring boot 核心依赖-->
4      <dependency>
5          <groupId>org.springframework.boot</groupId>
6          <artifactId>spring-boot-starter</artifactId>
7      </dependency>
8      <!--配置提示-->
9      <dependency>
10         <groupId>org.springframework.boot</groupId>
11         <artifactId>spring-boot-configuration-processor</artifactId>
12     </dependency>
13     <!-- minio -->
14     <dependency>
15         <groupId>io.minio</groupId>
16         <artifactId>minio</artifactId>
17     </dependency>
18     <!-- lombok -->
19     <dependency>
20         <groupId>org.projectlombok</groupId>
21         <artifactId>lombok</artifactId>
22     </dependency>
23 </dependencies>
```

3. 定义 Properties 配置文件

```
graph TD
    basic[basic] --> fzdxx[fzdkx-file-starter]
    fzdxx --> src[src]
    src --> main[main]
    main --> java[java]
    java --> com[com.fzdkx.file]
    com --> config[config]
    config --> MinIOAutoConfig[MinIOAutoConfig]
    config --> MinIOConfigProperties[MinIOConfigProperties]
```

The diagram illustrates the project structure. The 'basic' module contains the 'fzdkx-file-starter' module. Inside 'fzdkx-file-starter', there is a 'src' directory, which contains a 'main' directory. The 'main' directory contains a 'java' directory, which contains the package 'com.fzdkx.file'. Inside 'com.fzdkx.file', there is a 'config' directory. The 'config' directory contains two files: 'MinIOAutoConfig' and 'MinIOConfigProperties'. The 'MinIOConfigProperties' file is highlighted with a red box.

```
1  /**
2   * @author 发着呆看星
3   * @create 2024/2/5
4   */
5  @Data
6  @ConfigurationProperties(prefix = "minio")
7  public class MinIOConfigProperties {
8      // 账户
9      private String accessKey;
10     // 密码
11     private String secretKey;
12     // 桶
13     private String bucket;
14     // minio服务端 ip + port
15     private String endpoint;
16     // minio存储 ip + port
17     private String readPath;
18     // 这两个目前是一样的
19 }
```

4. 编写 minio 操作接口

service

impl

FileStorageServiceImpl

FileStorageService


```
1  /**
2   * @author 发着呆看星
3   * @create 2024/2/5
4   * minio操作接口
5   */
6  public interface FileStorageService {
7
8      /**
9       * 上传图片文件
10      * @param prefix 文件前缀
11      * @param filename 文件名
12      * @param inputStream 文件流
13      * @return 文件全路径
14      */
15      String uploadImgFile(String prefix, String filename, InputStream inputStream);
16
17      /**
18       * 上传html文件
19       * @param prefix 文件前缀
20       * @param filename 文件名
21       * @param inputStream 文件流
22       * @return 文件全路径
23       */
24      String uploadHtmlFile(String prefix, String filename, InputStream inputStream);
25
26      /**
27       * 删除文件
28       * @param pathUrl 文件全路径
29       */
30      void delete(String pathUrl);
31
32      /**
33       * 下载文件
34       * @param pathUrl 文件全路径
35       * @return 返回文件字节数组
36       */
37      byte[] downloadFile(String pathUrl);
38  }
```

5. 编写 minio 操作接口实现类，完成业务逻辑

```
1  /**
2   * @author 发着呆看星
3   * @create 2024/2/5
4   */
5  @Slf4j
6  @Data
7  public class FileStorageServiceImpl implements FileStorageService {
8
9      private MinioClient minioClient;
10
11     private MinIOConfigProperties properties;
12
13     private final String separator = "/";
14
15     /**
16      * @param dirPath 文件路径
17      * @param filename 文件名
18      * @return 完整路径 dirPath/ + yyyy/MM/dd/ + filename
19      */
20     public String builderFilePath(String dirPath, String filename) {
21         // 构建文件路径
22         // 创建StringBuilder
23         StringBuilder stringBuilder = new StringBuilder(50);
24         // 如果 dirPath 不为 null
25         if (StringUtils.hasLength(dirPath)) {
26             // 拼接路径
27             stringBuilder.append(dirPath).append(separator);
28         }
29         // 格式化日期 SimpleDateFormat
30         SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
31         // 使用当前时间, 进行日期格式化, 作为路径
32         String todayStr = sdf.format(new Date());
33         // dirPath/ + yyyy/MM/dd/ + filename
34         stringBuilder.append(todayStr).append(separator);
35         stringBuilder.append(filename);
36         return stringBuilder.toString();
37     }
38
39     /**
40      * 上传图片
41      *
42      * @param prefix 文件前缀
43      * @param filename 文件名
44      * @param inputStream 文件流
45      * @return 图片路径 readPath + /bucket + /filepath
```

```

46     */
47     @Override
48     public String uploadImgFile(String prefix, String filename,
49                               InputStream inputStream) {
50         // 构建文件路径
51         String filePath = builderFilePath(prefix, filename);
52         // 上传文件
53         try {
54             // 构建上传文件对象
55             PutObjectArgs putObjectArgs = PutObjectArgs.builder()
56                 .object(filePath) // 文件名
57                 .contentType("image/jpg") // 文件类型
58                 .bucket(properties.getBucket()) // 文件所在桶
59                 .stream(inputStream, inputStream.available(), -1) //
60             整合原文件
61                 .build();
62             return putObject(filePath, putObjectArgs);
63         } catch (Exception e) {
64             log.error("minio put file error.", e);
65             throw new RuntimeException("上传文件失败");
66         }
67     }
68     /**
69     * @param prefix    文件前缀
70     * @param filename   文件名
71     * @param inputStream 文件流
72     * @return html文件路径 readPath + /bucket + /filepath
73     */
74     @Override
75     public String uploadHtmlFile(String prefix, String filename,
76                                 InputStream inputStream) {
77         // 构建文件路径
78         String filePath = builderFilePath(prefix, filename);
79         try {
80             // 构建删除文件对象
81             PutObjectArgs putObjectArgs = PutObjectArgs.builder()
82                 .object(filePath)
83                 .contentType("text/html")
84                 .bucket(properties.getBucket())
85                 .stream(inputStream, inputStream.available(), -1)
86                 .build();
87             return putObject(filePath, putObjectArgs);
88         } catch (Exception e) {
89             log.error("minio put file error.", e);
90             throw new RuntimeException("上传文件失败");
91         }
92     }

```

```

93
94     @NotNull
95     private String putObject(String filePath, PutObjectArgs putObjectArgs
96 )
97         throws Exception{
98         minioClient.putObject(putObjectArgs);
99         StringBuilder urlPath = new StringBuilder(properties.getReadPath(
100 ));
101         urlPath.append(separator);
102         urlPath.append(properties.getBucket());
103         urlPath.append(separator);
104         urlPath.append(filePath);
105         return urlPath.toString();
106     }
107
108     @Override
109     public void delete(String pathUrl) {
110         // 将pathUrl中的 ip+port+/ 删除
111         String key = pathUrl.replace(properties.getEndpoint() + "/", "");
112         // 获取 / 第一次出现的位置 , 即为bucket后的 /
113         int index = key.indexOf(separator);
114         // 获取 bucket [ )
115         String bucket = key.substring(0, index);
116         // 获取文件路径, 不带 /
117         String filePath = key.substring(index + 1);
118         // 删除文件
119         // 构建删除文件对象
120         RemoveObjectArgs removeObjectArgs = RemoveObjectArgs.builder()
121             .bucket(bucket)
122             .object(filePath)
123             .build();
124         try {
125             minioClient.removeObject(removeObjectArgs);
126         } catch (Exception e) {
127             log.error("minio remove file error. pathUrl:{", pathUrl);
128             throw new RuntimeException("上传删除失败");
129         }
130     }
131
132     /**
133      * @param pathUrl 文件全路径
134      * @return 文件字节流
135      */
136     @Override
137     public byte[] downloadFile(String pathUrl) {
138         // 去除 ip + port + /
139         String key = pathUrl.replace(properties.getEndpoint() + "/", "");

```

```

139 // 获取第一个 /
140 int index = key.indexOf(separator);
141 // 获取bucket
142 String bucket = key.substring(0, index);
143 // 获取 filePath
144 String filePath = key.substring(index + 1);
145 // 下载文件
146 InputStream inputStream = null;
147 try {
148     inputStream = minioClient.getObject(
149         GetObjectArgs.builder()
150             .bucket(bucket)
151             .object(filePath)
152             .build()
153     );
154 } catch (Exception e) {
155     log.error("minio down file error. pathUrl:{", pathUrl);
156     throw new RuntimeException("上传下载失败");
157 }
158 // 将下载文件转换成 byte[]
159 ByteArrayOutputStream baos = new ByteArrayOutputStream();
160 byte[] bytes = new byte[100];
161 int i = 0;
162 while (true) {
163     try {
164         if (!((i = inputStream.read(bytes, 0, 100)) > 0)) {
165             break;
166         }
167     } catch (IOException e) {
168         e.printStackTrace();
169     }
170     baos.write(bytes, 0, i);
171 }
172 return baos.toByteArray();
173 }
174 }

```

6. 编写 starter 自动配置类

```
1  /**
2   * @author 发着呆看星
3   * @create 2024/2/5
4   */
5  @Configuration
6  @EnableConfigurationProperties({MinIOConfigProperties.class})
7  public class MinIOAutoConfig {
8      @Autowired
9      private MinIOConfigProperties properties;
10
11      @Bean
12      public MinioClient minioClient(){
13          return MinioClient
14              .builder()
15              .credentials(properties.getAccessKey(), properties.getSecretKey())
16              .endpoint(properties.getEndpoint())
17              .build();
18      }
19
20      @Bean
21      public FileStorageService fileStorageService
22          (MinIOConfigProperties minIOConfigProperties,
23          MinioClient minioClient){
24          FileStorageServiceImpl fileStorageService = new FileStorageService
25              Impl();
26          fileStorageService.setProperties(minIOConfigProperties);
27          fileStorageService.setMinioClient(minioClient);
28          return fileStorageService;
29      }
30  }
```

7. 利用 SpringBoot 中的 SPI 机制，完成自动注入

META-INF

- spring
 - org.springframework.boot.autoconfigure.AutoConfiguration.imports

▼ resources

▼ META-INF.spring

org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
1 com.fzdkx.file.config.MinIOAutoConfig
```

8. 其他模块导入该 starter

```
1 <dependency>
2   <groupId>com.fzdkx</groupId>
3   <artifactId>fzdkx-file-starter</artifactId>
4   <version>1.0</version>
5 </dependency>
```

9. 配置文件配置

```
1 minio:
2   accessKey: minio@123
3   secretKey: minio@123
4   bucket: leadnews
5   endpoint: http://192.168.16.166:9000
6   readPath: http://192.168.16.166:9000
```

10. 编写测试类测试

```
1  @SpringBootTest(classes = MinIoApplication.class)
2  @RunWith(SpringRunner.class)
3  public class FileStarterTest {
4      @Autowired
5      private FileStorageService fileStorageService;
6
7      @Test
8      public void testUpdateImgFile() {
9          try {
10              FileInputStream fileInputStream =
11                  new FileInputStream("D:\\武.webp");
12              String filePath =
13                  fileStorageService.uploadImgFile("", "武.webp", fileInputS
14                      tream);
15              System.out.println(filePath);
16          } catch (FileNotFoundException e) {
17              e.printStackTrace();
18          }
19      }
```

11. 测试结果

Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported
<http://192.168.16.166:9000/leadnews/2024/02/05/武.webp>



leadnews / 2024 / 02 / 05



^ Name



武.webp