

多活高可用服务

开发指南

文档版本 09
发布日期 2025-07-22



版权所有 © 华为云计算技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址：贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址：<https://www.huaweicloud.com/>

目录

1 MAS-DB-SDK 使用手册	1
1.1 概述.....	1
1.1.1 开发简介.....	1
1.1.2 开发流程.....	2
1.2 约束.....	2
1.2.1 环境约束.....	3
1.2.2 版本依赖.....	3
1.2.3 多活容灾监控准备.....	4
1.3 接入指南.....	5
1.3.1 SpringBoot 项目接入 MAS-DB-SDK.....	5
1.3.2 Spring 项目接入 MAS-DB-SDK.....	7
1.4 使用场景.....	9
1.4.1 单边读写.....	9
1.4.1.1 无从库.....	10
1.4.1.2 读写分离.....	11
1.4.2 本地读单边写.....	13
1.4.2.1 无从库.....	13
1.4.2.2 读写分离.....	14
1.4.3 强制路由.....	16
1.4.3.1 非事务中使用强制路由.....	17
1.4.3.2 事务中使用强制路由.....	17
1.4.4 使用 ShardingSphereDataSource 场景.....	18
1.4.5 使用多数据源.....	20
1.5 参数配置说明.....	22
1.5.1 配置项详细信息列表.....	22
1.6 不支持的接口.....	26
1.7 常见问题.....	30
1.7.1 如何选择组件版本.....	30
1.7.2 props、etcd 配置项在 MAS 服务中如何查询.....	30
1.7.3 文档中关于 SDK 对应 yaml 配置文件的获取方式.....	31
1.7.4 运行 MAS-DB-SDK 问题参考.....	32
2 MAS-Redis-SDK 使用手册	41
2.1 概述.....	41

2.1.1 开发简介.....	41
2.1.2 开发流程.....	42
2.2 约束.....	42
2.2.1 环境约束.....	42
2.2.2 版本依赖.....	42
2.2.3 多活容灾 Redis 监控准备.....	45
2.3 使用场景.....	45
2.3.1 single-read-write (单边读写)	45
2.3.2 local-read-single-write (本地读单边写)	46
2.3.3 local-read-async-double-write (本地读异步双写)	46
2.3.4 single-read-async-double-write (单边读异步双写)	47
2.4 接入指南.....	48
2.4.1 Spring 项目接入 MAS-Redis-SDK.....	48
2.4.2 SpringBoot 项目接入 MAS-Redis-SDK.....	50
2.4.3 Spring 项目接入 MAS-Redis-SDK (单实例)	51
2.4.4 SpringBoot 项目接入 MAS-Redis-SDK (单实例)	52
2.4.5 多客户端场景.....	53
2.5 命令参考.....	53
2.5.1 Redis 命令参考.....	53
2.5.2 pipeline 功能使用示例.....	59
2.5.3 DcsConnetion 命令参考.....	59
2.5.4 RedisTemplate 命令与 DcsConnection 接口对应关系.....	60
2.6 参数配置说明.....	64
2.7 敏感信息加解密.....	71
2.8 客户各场景替换方案.....	71
2.8.1 Jedis、Lettuce.....	71
2.8.2 Spring-Boot-Data-Redis.....	72
2.8.3 ShedLock.....	73
2.8.4 Redisson Lock.....	73
2.8.5 自定义 DcsConnection.....	75
2.9 分布式锁场景最佳实践.....	76
3 MAS-ElasticSearch-SDK 使用手册.....	79
3.1 概述.....	79
3.2 多活容灾 ElasticSearch 监控准备.....	79
3.3 使用场景.....	79
3.3.1 local-read-write (单边读写)	80
3.4 接入指南.....	80
3.4.1 SpringBoot 项目接入 MAS-ElasticSearch-SDK.....	80
3.5 SingleReadWriteClient 命令参考.....	81
3.6 参数配置说明.....	81
4 MAS-Mongo-SDK 使用手册.....	83
4.1 概述.....	83

4.1.1 开发简介.....	83
4.1.2 开发流程.....	83
4.2 约束.....	84
4.2.1 版本约束.....	84
4.2.2 多活容灾 MongoDB 监控准备.....	84
4.3 接入指南.....	84
4.3.1 Spring 项目接入 MAS-Mongo-SDK.....	84
4.3.2 SpringBoot 项目接入 MAS-Mongo-SDK.....	86
4.4 使用场景.....	87
4.4.1 单边读写.....	87
4.4.2 本地读单边写.....	88
4.4.3 强制路由.....	89
4.4.4 读命令列表.....	90
4.5 配置参数说明.....	90
4.6 常见问题.....	94
4.6.1 如何选择组件版本.....	94
4.6.2 使用限制.....	95
5 MAS-GO-SDK 使用手册.....	96
5.1 MySQL.....	96
5.1.1 概述.....	96
5.1.1.1 开发简介.....	96
5.1.1.2 开发流程.....	96
5.1.2 环境准备.....	97
5.1.3 使用场景.....	97
5.1.4 使用指南.....	98
5.1.4.1 原生 DB.....	98
5.1.4.2 beego-orm.....	101
5.1.4.3 gorm.....	102
5.1.5 配置项说明.....	103
5.2 Redis.....	105
5.2.1 概述.....	105
5.2.1.1 开发简介.....	106
5.2.1.2 开发流程.....	106
5.2.2 环境准备.....	106
5.2.3 使用场景.....	106
5.2.4 使用指南.....	108
5.2.5 配置项说明.....	111
5.3 故障注入.....	114
5.3.1 概述.....	114
5.3.2 使用指南.....	114
5.3.2.1 MySQL 配置示例.....	114
5.3.2.2 Redis 配置示例.....	117

5.3.3 配置项说明.....	119
5.3.4 内置注入故障.....	120
6 MAS-SDK 版本变更记录下载.....	121

1 MAS-DB-SDK 使用手册

- 1.1 概述
- 1.2 约束
- 1.3 接入指南
- 1.4 使用场景
- 1.5 参数配置说明
- 1.6 不支持的接口
- 1.7 常见问题

1.1 概述

1.1.1 开发简介

本文主要描述如何使用MAS-DB-SDK在多活容灾场景下对涉及配置多数据源服务进行开发，结合样例讲解MAS-DB-SDK在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Java语言，并有Java程序开发经验。
- 熟悉Maven。
- 熟悉服务配置数据源方式。

MAS-DB-SDK

MAS-DB-SDK是一个DataSource的实现，可替代spring-boot-starter-jdbc为Spring注入一个DataSource对象，通过与MAS服务对接从而实现多数据源管理的工具，具有以下特性：

- 多活容灾能力。
多活容灾能力是指在同城场景下实现多活故障自动切换，由SDK和MAS服务配合完成。

- 读写分离。
读写分离由SDK实现，支持随机、轮询的负载均衡算法。
- 自定义指定数据源进行数据访问。
用户也可以通过注解方式指定数据源以及主从数据库进行数据读写。

```
@DynamicRoute(source = "dc1", hint = HintType.READSLAVE)
@RequestMapping(value = "/insert2", method = RequestMethod.POST)
public int insert2(@RequestBody UserModel user) {
    int ret = userMapper.insert(user);
    return ret;
}
```

注解中source是指定选择哪个节点的数据源，从YAML配置中的router.nodes中选取，如果不设置，请使用当前DCG仲裁的数据中心（MAS平台上激活的数据中心，如果未对接MAS平台，则选取本地配置中的active，即yaml配置中的router.active）。

说明

DCG为多活实例的仲裁中心，提供端到端（流量-应用-数据）仲裁和统一多活切换管控能力。

注解中的HintType用于查询时指定主从数据库，适用于读写分离场景，当前支持的配置值有：HintType.READMASTER（从master库读取）、HintType.READSLAVE（从slave库读取）、HintType.NONE（不指定），不指定时，默认从slave库读取。HintType只针对读操作有效，写操作HintType无效。

须知

SDK本身不支持数据源间的数据同步，数据源同步需要依赖DRS服务。
SDK不支持分布式事务，同一个事务处理过程中，不支持切换数据库。

1.1.2 开发流程

开发的流程如下所示：

1. 版本获取及引入依赖。
通过Maven引入需要的依赖，是使用MAS-DB-SDK的基础。
2. 添加客户端配置。
通过添加客户端配置，接入MAS-DB-SDK。
3. 创建DataSource。
MAS-DB-SDK提供读取YAML文件创建DataSource的方法，如果是springboot项目，可以通过使用starter自动注入DataSource，详见[接入指南](#)。

1.2 约束

1.2.1 环境约束

安装的工具包括JDK、Maven、IDEA，配置对应的环境变量，确保本地开发环境可用。

📖 说明

使用MAS-DB-SDK组件需具备一定的Java后端、持久层框架集成等知识。

准备项	说明
准备操作系统	Windows系统。Windows版本要求：Windows 7及以上版本。
安装JDK	开发环境的基本配置。JDK版本要求：1.8.0_262及以上版本。
安装Maven	MAS-DB-SDK使用Maven获取项目版本。Maven版本要求：3.3.0及以上版本。
安装和配置IntelliJ IDEA	用于开发程序的工具。建议IntelliJ IDEA版本为2020及以上版本。

1.2.2 版本依赖

下面是MAS-DB-SDK中引入的依赖及依赖版本：

所属模块	依赖名称	依赖版本	scope
devspore-datasource	com.huaweicloud.devspore:devspore-mas-common	latest	compile
	net.jodah:failsafe	2.4.0	compile
	com.github.jsqlparser:jsqlparser	4.2	compile
	com.fasterxml.jackson.core:jackson-core	2.13.2	compile
	com.fasterxml.jackson.core:jackson-databind	2.13.2.2	compile
	com.fasterxml.jackson.dataformat:jackson-dataformat-yaml	2.13.2	compile
	org.apache.commons:commons-lang3	3.12.0	compile
	commons-io:commons-io:jar	2.11.0	compile

所属模块	依赖名称	依赖版本	scope
	com.google.guava:guava:jar	30.1.1-jre	compile
	com.zaxxer:HikariCP	4.0.3	optional
	org.apache.commons:commons-dbcp2	2.8.0	optional
	com.alibaba:druid	1.2.9	optional
	org.projectlombok:lombok	1.18.20	provided
	org.springframework:spring-aspects	5.3.9	compile
	org.codehaus.groovy	3.0.8	compile
devspore-mas-common	org.slf4j:slf4j-api	1.7.36	compile
	io.etcd:jetcd-core	0.5.7	compile
	com.google.errorprone:error_prone_annotations	2.5.1	compile
spring-cloud-starter-huawei-devspore-datasource	com.huaweicloud.devspore:devspore-datasource	latest	compile
	org.springframework.boot:spring-boot-autoconfigure	2.5.4	compile
	org.projectlombok:lombok	1.18.20	provided
	org.springframework.boot:spring-boot-autoconfigure-processor	2.5.4	compile
	org.springframework.boot:spring-boot-configuration-processor	2.5.4	compile
	jakarta.annotation:jakarta.annotation-api	2.0.0	compile

1.2.3 多活容灾监控准备

在使用MAS-DB-SDK进行开发前，需要先做好如下准备：

1. 已创建MAS实例。
2. 在实例下已创建对应的监控器。

请参考[MySQL/Oracle/PostgreSQL监控管理](#)，配置多活容灾MySQL/Oracle/PostgreSQL监控。

1.3 接入指南

1.3.1 SpringBoot 项目接入 MAS-DB-SDK

步骤1 引入MAS-DB-SDK组件坐标，下面version版本为最新版本，依赖组件版本参考[如何选择组件版本](#)。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>spring-cloud-starter-huawei-devspore-datasource</artifactId>
  <version>1.2.1-RELEASE</version>
</dependency>
```

步骤2 根据本项目配置项选其中一个数据连接池坐标。

```
<dependency>
  <groupId>com.zaxxer</groupId>
  <artifactId>HikariCP</artifactId>
  <version>4.0.3</version>
</dependency>

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
  <version>2.8.0</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.2.6</version>
</dependency>
```

步骤3 项目配置文件application.yaml或者application.properties中加入MAS-DB-SDK配置项。

须知

- 下文中配置项的格式为yaml文件，如果配置文件为properties格式，需自行修改格式。
 - MAS-DB-SDK配置项分为4部分：props配置、etcd配置、sources数据源配置、router路由配置。
 - 如果使用MAS服务，props配置、etcd配置则必须要配置，sources数据源配置需要与MAS服务中连接池中的命名一致。如果不对接MAS服务，props配置、etcd配置无需配置，sources数据源配置本地所用数据库即可。
 - 下列场景中的配置都包含了props、etcd的配置，配置项的具体值含义参考[配置项详细信息列表](#)。
-
- MAS-DB-SDK配置示例，这里的配置只作为一个格式的展示，具体场景的具体配置参考章节[配置说明](#)。

```
devspore:
datasource:
  props:
    version: v1
    appld: xxx
    monitorId: xxx
    databaseName: xxx
    decipherClassName: xxx.xxx.xxx
    region: az0
  etcd:
    address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379
    apiVersion: v3
    username: etcduser
    password: etcdpwd
    httpsEnable: false
  sources:
    ds1:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
        maximumPoolSize: 200
    ds1-slave0:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave0
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
        maximumPoolSize: 200
    ds1-slave1:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave1
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
        maximumPoolSize: 200
    ds2:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
        maximumPoolSize: 200
    ds2-slave0:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave0
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
        maximumPoolSize: 200
    ds2-slave1:
      driverClassName: com.mysql.jdbc.Driver
      jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave1
      username: datasourceuser
      password: datasourcepwd
      type: com.zaxxer.hikari.HikariDataSource
      props:
        connectionTimeout: 30000
```

```
maximumPoolSize: 200
router:
  active: dc1
  routeAlgorithm: single-read-write
  nodes:
    dc1:
      master: ds1
      loadBalance: ROUND_ROBIN
      slaves:
        - ds1-slave0
        - ds1-slave1
    dc2:
      master: ds1
      loadBalance: ROUND_ROBIN
      slaves:
        - ds2-slave0
        - ds2-slave1
```

步骤4 使用MAS-DB-SDK提供注解在类级别、方法级别指定数据源。

📖 说明

- 此步骤为可选步骤，如果在本项目中使用不到，忽略此步骤即可。
- 如果不指定数据源，默认使用application.yaml、application.properties中router下配置的active节点。
- 类级别指定数据源示例：

见[1.4.3 强制路由](#)



```
DatasourceDemoApplication.java × pom.xml (datasource-demo) × TestUserController.java × application.yaml ×
1 package com.huawei.control;
2
3
4 import ...
14
15
16 @RestController
17 @RequestMapping(value = "/v1/user", produces = {"application/json;charset=UTF-8"})
18 @DynamicRoute(source = "dc2", hint = HintType.READMASTER)
19 public class TestUserController {
20     @Autowired
21     private UserMapper userMapper;
22
23     @RequestMapping(value = "/insert", method = RequestMethod.POST)
24     public int insert(@RequestBody UserModel user) {
25         int ret = userMapper.insert(user);
26         return ret;
27     }
28 }
```

- 方法级别指定数据源示例：

见[1.4.3 强制路由](#)

----结束

1.3.2 Spring 项目接入 MAS-DB-SDK

步骤1 引入MAS-DB-SDK组件坐标，下面version版本为最新版本，依赖组件版本参考[如何选择组件版本](#)。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>devspore-datasource</artifactId>
  <version>1.2.1-RELEASE</version>
</dependency>
```

步骤2 根据本项目配置项选其中一个数据连接池坐标。

```
<dependency>
  <groupId>com.zaxxer</groupId>
  <artifactId>HikariCP</artifactId>
  <version>4.0.3</version>
</dependency>

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
  <version>2.8.0</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.2.6</version>
</dependency>
```

步骤3 增加一个配置yml或properties配置文件，引入MAS-DB-SDK配置项。**说明**

- 下文中配置项的格式为yaml文件，如果配置文件为properties格式，需自行修改格式。
- MAS-DB-SDK配置项分为4部分：props配置、etcd配置、sources数据源配置、router路由配置。
- 如果使用MAS服务，props配置、etcd配置则必须要配置，sources数据源配置需要与MAS服务中连接池中的命名一致。如果不对接MAS服务，props配置、etcd配置无需配置，sources数据源配置本地所用数据库即可。
- 下列场景中的配置都包含了props、etcd的配置，配置项的具体值含义参考[配置项详细信息列表](#)。
- MAS-DB-SDK配置示例，这里的配置只作为一个格式的展示，具体场景的具体配置参考章节[配置说明](#)。

```
props:
  version: v1
  appld: xxx
  monitorId: xxx
  databaseName: xxx
  decipherClassName: xxx.xxx.xxx

etcd:
  address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379
  apiVersion: v3
  username: etcduser
  password: etcdpwd
  httpsEnable: false

sources:
  ds1:
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connection-timeout: 1000
      validation-timeout: 1000
  ds1-slave0:
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave0
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
  ds1-slave1:
    driverClassName: com.mysql.jdbc.Driver
```

```
jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave1
username: datasourceuser
password: datasourcepwd
type: com.zaxxer.hikari.HikariDataSource
ds2:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
ds2-slave0:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave0
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
ds2-slave1:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource

router:
  active: dc1
  routeAlgorithm: single-read-write
  nodes:
    dc1:
      master: ds1
      loadBalance: ROUND_ROBIN
      slaves:
        - ds1-slave0
        - ds1-slave1
    dc2:
      master: ds2
      loadBalance: round_robin
      slaves:
        - ds2-slave0
        - ds2-slave1
```

步骤4 创建DataSource的Bean。

```
@Bean
public DataSource dataSource() {
    DataSource clusterDataSource= YamlClusterDataSourceFactory.createDataSource("D:\\Coral\\Mas
\\TestService\\src\\main\\resources\\devspore-datasource.yml");
    return clusterDataSource;
}
```

步骤5 （可选）使用MAS-DB-SDK提供注解在类级别、方法级别指定数据源参考[步骤4](#)。

----结束

1.4 使用场景

1.4.1 单边读写

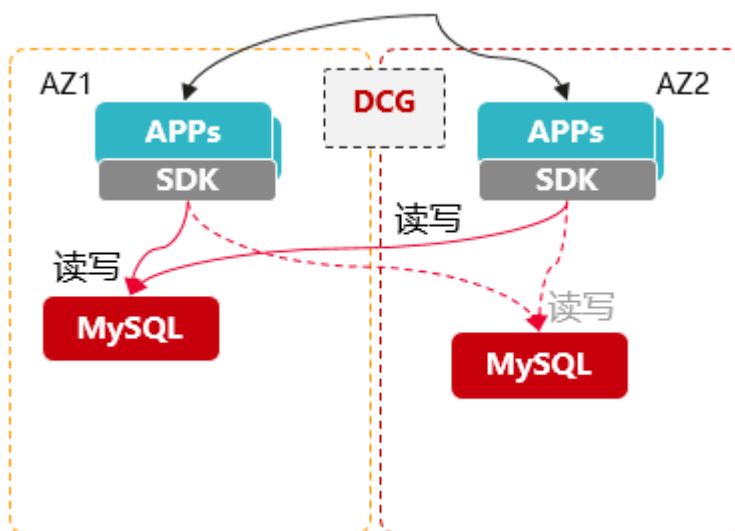
这种路由算法场景下，对数据库的读写操作都只在一边（数据中心1或者数据中心2）进行，根据active指示，在相应的那边数据中心操作。（通过注解指定的场景例外，注解指定见[强制路由](#)）

须知

路由算法不支持动态切换，更改路由算法，需要重启服务。

1.4.1.1 无从库

场景一：



如上图，实线为DCG指示激活AZ1，虚线为DCG指示激活为AZ2。DCG指示当前激活的数据中心，SDK根据DCG的指示，在相应的数据中心操作，读写操作都在一边进行。

配置如下：

```
# 基础信息 - 可选项, 当配置etcd后为必选
props:
  version: v1 // 项目版本号, 自定义
  appld: xxx // 应用ID, 从MAS服务实例页面查询获取
  monitorId: xxx // 监控器ID, 从MAS服务实例页面查询获取
  databaseName: xxx // 数据库名, 从MAS服务实例页面查询获取
  decipherClassName: xxx.xxx.xxx // 加解密类, 需要实现基类 com.huawei.devspore.mas.password.Decipher,
  默认值为com.huawei.devspore.mas.password.DefaultDecipher

# etcd配置, 对接MAS服务关键配置, 本地模式则无需配置
etcd:
  address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379 //etcd地址, 从MAS服务实例页面查询获取
  apiVersion: v3 // etcd版本, v3
  username: etcduser // etcd用户名, 从MAS服务实例页面查询获取
  password: etcdpwd // etcd密码, 从MAS服务实例页面查询获取
  httpsEnable: false // 是否启用https
  certificatePath: xxx // 启用https时证书路径, 实例未开启双向认证或不启用https场景, 该配置可以不填

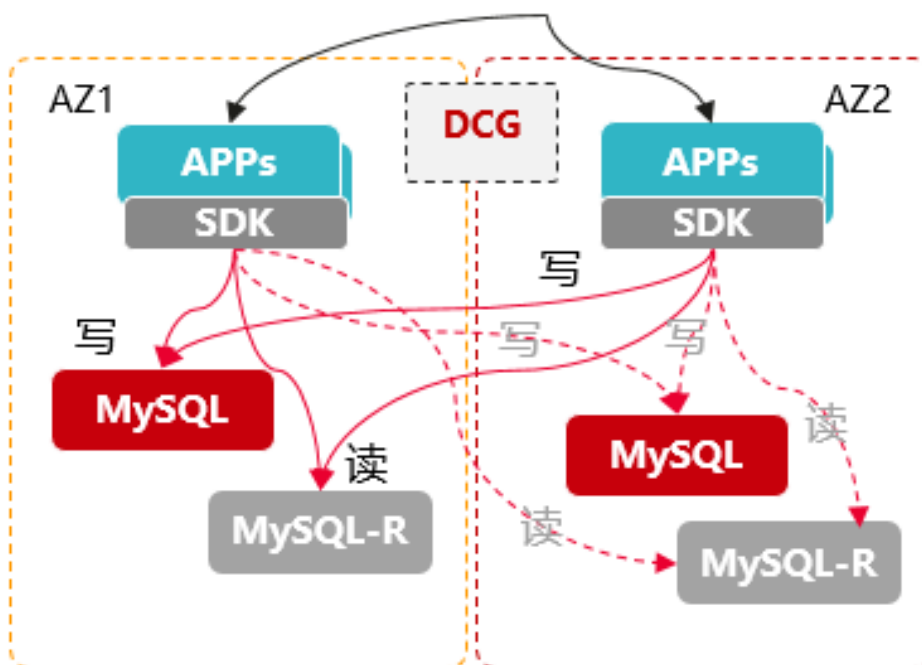
# 数据源配置 - 必选
sources:
  ds1: // 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver // 驱动名称, 自定义
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0 // 数据库地址, 和MAS服务配置一致
    username: datasourceuser // 用户名, 和MAS服务配置一致
    password: datasourcepwd // 密码, 和MAS服务配置一致
    type: com.zaxxer.hikari.HikariDataSource // 数据源类型, 自定义, 目前只支持,
    com.zaxxer.hikari.HikariDataSource,org.apache.commons.dbcp2.BasicDataSource,com.alibaba.druid.pool.DruidDataSource
  ds2: // 需要和MAS服务中连接池中命名一致
```

```
driverClassName: com.mysql.jdbc.Driver
jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
username: datasourceuser
password: datasourcepwd
type: com.zaxxer.hikari.HikariDataSource

# 路由配置 - 必选
router:
  active: dc1 // 当前激活节点
  routeAlgorithm: single-read-write // 路由策略
  nodes:
    dc1:
      master: ds1
    dc2:
      master: ds2
```

1.4.1.2 读写分离

场景二：



如上图，实线为DCG指示激活AZ1，虚线为DCG指示激活为AZ2。跟场景一类似，同样在单边进行读写，不同之处在于，场景二中存在从库，进行读写分离，写操作在主库，读操作在从库进行。

配置如下：

```
# 基础信息 - 可选项, 当配置etcd后为必选
props:
  version: v1 // 项目版本号, 自定义
  appld: xxx // 应用ID, 从MAS服务实例页面查询获取
  monitorId: xxx // 监控器ID, 从MAS服务实例页面查询获取
  databaseName: xxx // 数据库名, 从MAS服务实例页面查询获取
  decipherClassName: xxx.xxx.xxx // 加解密类, 需要实现基类 com.huawei.devspore.mas.password.Decipher,
  默认值为com.huawei.devspore.mas.password.DefaultDecipher

# etcd配置, 对接MAS服务关键配置, 本地模式则无需配置
etcd:
  address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379 //etcd地址, 从MAS服务实例页面查询获取
```

```
apiVersion: v3 // etcd版本, v3
username: etcduser // etcd用户名, 从MAS服务实例页面查询获取
password: etcdpwd // etcd密码, 从MAS服务实例页面查询获取
httpsEnable: false // 是否启用https
certificatePath: xxx // 启用https时证书路径, 实例未开启双向认证或不启用https场景, 该配置可以不填

# 数据源配置 - 必选
sources:
  ds1: // 数据库直接点配置, 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver // 驱动名称, 自定义
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0 // 数据库地址, 和MAS服务配置一致
    username: datasourceuser // 用户名, 和MAS服务配置一致
    password: datasourcepwd // 密码, 和MAS服务配置一致
    type: com.zaxxer.hikari.HikariDataSource // 数据源类型, 自定义, 目前只支持
    com.zaxxer.hikari.HikariDataSource,org.apache.commons.dbcp2.BasicDataSource,com.alibaba.druid.pool.DruidDataSource
  ds1-slave0:// 从数据库节点,需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave0
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds1-slave1:
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave1
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds2: // 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds2-slave0:// 从数据库节点,需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave0
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds2-slave1:
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave1
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200

# 路由配置 - 必选
router:
  active: dc1
  routeAlgorithm: single-read-write
  nodes:
    dc1:
      master: ds1
```


配置如下:

```
# 基础信息 - 可选项, 当配置etcd后为必选
props:
  version: v1 // 项目版本号, 自定义
  appld: xxx // 应用ID, 从MAS服务实例页面查询获取
  monitorId: xxx // 监控器ID, 从MAS服务实例页面查询获取
  dbName: xxx // 数据库名, 从MAS服务实例页面查询获取
  decipherClassName: xxx.xxx.xxx // 加解密类, 需要实现基类 com.huawei.devspore.mas.password.Decipher,
  默认值为com.huawei.devspore.mas.password.DefaultDecipher
  azs: az1 // 本地的AZ信息, 根据实际环境所属AZ进行填写

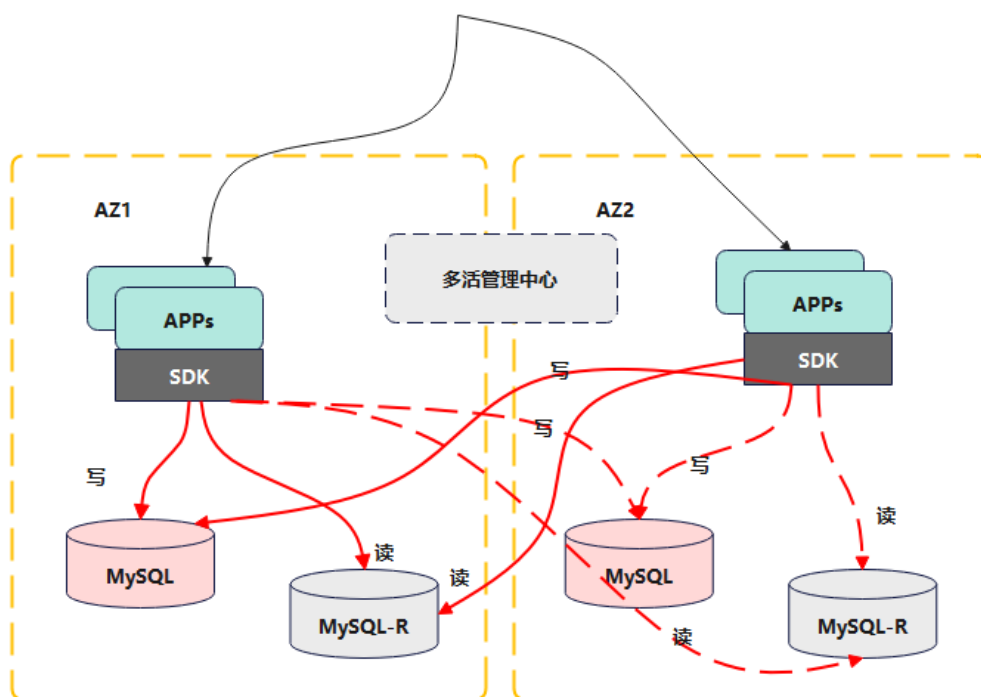
# etcd配置, 对接MAS服务关键配置, 本地模式则无需配置
etcd:
  address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379 //etcd地址, 从MAS服务实例页面查询获取
  apiVersion: v3 // etcd版本, v3
  username: etcduser // etcd用户名, 从MAS服务实例页面查询获取
  password: etcdpwd // etcd密码, 从MAS服务实例页面查询获取
  httpsEnable: false // 是否启用https
  certificatePath: xxx // 启用https时证书路径, 实例未开启双向认证或不启用https场景, 该配置可以不填

# 数据源配置 - 必选
sources:
  ds1: // 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver // 驱动名称, 自定义
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0 // 数据库地址, 和MAS服务配置一致
    username: datasourceuser // 用户名, 和MAS服务配置一致
    password: datasourcepwd // 密码, 和MAS服务配置一致
    type: com.zaxxer.hikari.HikariDataSource
    props: // 连接池参数
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds2: // 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
    username: datasourceuser
    password: datasourcepwd
    type: com.zaxxer.hikari.HikariDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200

# 路由配置 - 必选
router:
  active: dc1 // 当前激活节点
  routeAlgorithm: local-read-single-write // 路由策略
  nodes:
    dc1:
      master: ds1
      azs: az1
    dc2:
      master: ds2
      azs: az2
```

1.4.2.2 读写分离

场景四:



如上图，实线为DCG指示激活AZ1，虚线为DCG指示激活为AZ2。跟场景三不同之处在于，场景四中存在从库，进行读写分离，写操作在主库，读操作在从库进行。

配置如下：

```
# 基础信息 - 可选项, 当配置etcd后为必选
props:
  version: v1 // 项目版本号, 自定义
  appld: xxx // 应用ID, 从MAS服务实例页面查询获取
  monitorId: xxx // 监控器ID, 从MAS服务实例页面查询获取
  databaseName: xxx // 数据库名, 从MAS服务实例页面查询获取
  decipherClassName: xxx.xxx.xxx // 加解密类, 需要实现基类 com.huawei.devspore.mas.password.Decipher,
  默认值为com.huawei.devspore.mas.password.DefaultDecipher
  azs: az1

# etcd配置, 对接MAS服务关键配置, 本地模式则无需配置
etcd:
  address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379 //etcd地址, 从MAS服务实例页面查询获取
  apiVersion: v3 // etcd版本, v3
  username: etcduser // etcd用户名, 从MAS服务实例页面查询获取
  password: etcdpwd // etcd密码, 从MAS服务实例页面查询获取
  httpsEnable: false // 是否启用https
  certificatePath: xxx // 启用https时证书路径, 实例未开启双向认证或不启用https场景, 该配置可以不填

# 数据源配置 - 必选
sources:
  ds1: // 数据库直接点配置, 需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver // 驱动名称, 自定义
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0 // 数据库地址, 和MAS服务配置一致
    username: datasourceuser // 用户名, 和MAS服务配置一致
    password: datasourcepwd // 密码, 和MAS服务配置一致
    type: com.zaxxer.hikari.HikariDataSource // 数据源类型, 自定义, 目前只支持
    com.zaxxer.hikari.HikariDataSource,org.apache.commons.dbcp2.BasicDataSource,com.alibaba.druid.pool.DruidDataSource
    props:
      connectionTimeout: 30000
      maximumPoolSize: 200
  ds1-slave0:// 从数据库节点,需要和MAS服务中连接池中命名一致
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave0
```

```
username: datasourceuser
password: datasourcepwd
type: com.zaxxer.hikari.HikariDataSource
props:
  connectionTimeout: 30000
  maximumPoolSize: 200
ds1-slave1:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0_slave1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
  props:
    connectionTimeout: 30000
    maximumPoolSize: 200
ds2: // 需要和MAS服务中连接池中命名一致
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
  props:
    connectionTimeout: 30000
    maximumPoolSize: 200
ds2-slave0:// 从数据库节点,需要和MAS服务中连接池中命名一致
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave0
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
  props:
    connectionTimeout: 30000
    maximumPoolSize: 200
ds2-slave1:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1_slave1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource
  props:
    connectionTimeout: 30000
    maximumPoolSize: 200

# 路由配置 - 必选
router:
  active: dc1
  routeAlgorithm: local-read-single-write
  nodes:
    dc1:
      master: ds1
      loadBalance: ROUND_ROBIN
      slaves:
        - ds1-slave0
        - ds1-slave1
      azs: az1
    dc2:
      master: ds2
      loadBalance: ROUND_ROBIN
      slaves:
        - ds2-slave0
        - ds2-slave1
      azs: az2
```

1.4.3 强制路由

1.4.3.1 非事务中使用强制路由

除了根据路由算法配置路由，SDK还支持通过注解**@DynamicRoute**强制指定路由，且相比路由算法，注解指定路由优先级更高。存在注解指定路由的场景下，优先根据注解指定进行路由。

注解**@DynamicRoute**指定路由分为两部分：source和hint，可单独只指定source或hint，也可一起组合使用。

source用于选择数据中心，可以填“dc1”/“dc2”/“ACTIVE”。填“dc1”时，则强制路由到dc1数据中心，不管此时的路由算法为哪种，也不管sql类型是读还是写；同理，填“dc2”时，强制路由到dc2数据中心；比较特殊的是“ACTIVE”，此时路由到active指示的数据中心，同样不管路由算法为哪种，也不管sql类型为哪种。

hint用于读写分离场景，可以填HintType.READMASTER/HintType.READSLAVE/HintType.NONE。其中，HintType.READMASTER为指定从主库读取，HintType.READSLAVE为指定从从库读取，HintType.NONE为不指定。当存在从库，且从库可读（DCG指示从库状态可读，若不对接MAS平台或DCG不指示，默认从库可读）时，默认从从库读取。

用法如下：

@DynamicRoute注解支持在方法、类上添加，如下：

方法上添加注解：

```
@DynamicRoute(source = "dc2", hint = HintType.READMASTER)
@RequestMapping(value = "/query/{id}", method = RequestMethod.GET)
public UserModel query(@PathVariable("id") Long id) {
    return userMapper.select(id);
}
```

类上添加注解：此时针对该类的所有方法注解都生效。

```
@DynamicRoute(source = "dc2", hint = HintType.READMASTER)
@RestController
@RequestMapping(value = "/v1/user", produces = {"application/json;charset=UTF-8"})
public class TestUserController {
    @Autowired
    private UserMapper userMapper;

    @RequestMapping(value = "/query/{id}", method = RequestMethod.GET)
    public UserModel query(@PathVariable("id") Long id) {
        return userMapper.select(id);
    }
}
```

1.4.3.2 事务中使用强制路由

SDK不支持分布式事务，事务与强制指定路由**@DynamicRoute**同时使用时，以第一条sql选择的数据源为准。

后续sql，即使使用**@DynamicRoute**指定数据源，也不生效。同一个事务中，不允许访问不同的数据库。

如下，service层的updateUser方法，开启了事务，并且里面有两个接口，第一个userReopository2上指定路由“dc2”，第二个userReopository2上指定路由“dc1”，此时以第一个接口中的注解指定的路由为准，使用“dc2”的数据源。

```
@Slf4j
@Repository
@javax.annotation.Generated(value = "com.huaweicloud.devspore.codegen, 1.5.1-SNAPSHOT")
@DynamicRoute(source = "dc1")
```

```
public class UserRepository extends AbstractUserRepository {

    public User update(User user) {
        return this.updateById(user);
    }
}

@Slf4j
@Repository
@javax.annotation.Generated(value = "com.huaweicloud.devspore.codegen, 1.5.1-SNAPSHOT")
@DynamicRoute(source = "dc2")
public class UserRepository2 extends AbstractUserRepository {

    public User update(User user) {
        return this.updateById(user);
    }
}

/**
 * UserService Class
 *
 * @since 2022-08-22
 */
@Slf4j
@org.springframework.stereotype.Service("UserService")
@javax.annotation.Generated(value = "com.huaweicloud.devspore.codegen, 1.5.1-SNAPSHOT")
public class UserService implements IUserService {
    // auto generated code

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private UserRepository2 userRepository2;

    /**
     * UpdateUserById Method
     *
     * @param user user
     * @return User
     */
    @Transactional
    public User updateUserById(User user) {
        userRepository2.update(user);
        return userRepository.update(user);
    }
}
```

1.4.4 使用 ShardingSphereDataSource 场景

SDK还支持使用ShardingSphereDataSource，此时，需要修改yml配置，单独将sharding配置放在yml同级目录，并在yml中设置sharding配置文件名，示例如下：

引入依赖：

```
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>5.0.0</version>
</dependency>
```

配置示例：

```
# 数据源配置 - 必选
sources:
  ds1:
```

```
type: org.apache.shardingsphere.driver.jdbc.core.datasource.ShardingSphereDataSource // 数据源类型
yamlFile: sharding.yml
ds2:
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://127.0.0.1:8080/ds1
  username: datasourceuser
  password: datasourcepwd
  type: com.zaxxer.hikari.HikariDataSource

# 路由配置 - 必选
router:
  active: dc1 // 当前激活节点
  routeAlgorithm: single-read-write // 路由策略
  nodes:
    dc1:
      master: ds1
    dc2:
      master: ds2
```

📖 说明

上面配置，仅ds1使用了ShardingSphereDataSource，ds2仍使用HikariDataSource，如果ds2也需要使用ShardingSphereDataSource，将ds2的type和yamlFile同样配置即可。

sharding配置参考shardingsphere官网，下面给出读写分离和分库分表的例子：

分库分表：

```
schemaName: ms
dataSources:
  ds0:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://100.100.139.118:3307/devspore-ds0
    username:
    password:
  ds1:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://100.100.139.118:3307/devspore-ds1
    username:
    password:
rules:
- !SHARDING
  bindingTables:
  - t_order,t_order_detail
  broadcastTables:
  - t_login
  - t_address
  - t_shopping_cart
  - t_bank
  tables:
    t_order_detail:
      actualDataNodes: ds$->{0..1}.t_order_detail_${0..2}
      tableStrategy:
        standard:
          shardingAlgorithmName: mod
          shardingColumn: id
      databaseStrategy:
        standard:
          shardingColumn: guest_id
          shardingAlgorithmName: mod
    t_order:
      actualDataNodes: ds$->{0..1}.t_order_${0..2}
      tableStrategy:
        standard:
          shardingAlgorithmName: mod
          shardingColumn: id
```

```
databaseStrategy:
  standard:
    shardingColumn: guest_id
    shardingAlgorithmName: mod

shardingAlgorithms:
  mod:
    type: MOD
    props:
      sharding-count: 2

props:
  sql-show: true
```

📖 说明

- 上面配置数据库用户名和密码未填，用户按需补充。
- 上面提供的配置只是一个例子，实际配置用户按自己的分库分表需求进行配置，这部分配置跟shardingsphere本身的分库分表配置一致。

读写分离：

```
schemaName: ms
dataSources:
  master:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://100.100.139.118:3307/devspore-ds0
    username:
    password:
  slave0:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://100.100.139.118:3307/devspore-ds0-slave0
    username:
    password:
  slave1:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://100.100.139.118:3307/devspore-ds0-slave1
    username: coral
    password: Coral

rules:
- !READWRITE_SPLITTING
  dataSources:
    ms_ds:
      writeDataSourceName: master
      readDataSourceNames:
        - slave0
        - slave1
      loadBalancerName: roundRobin

  loadBalancers:
    roundRobin:
      type: ROUND_ROBIN

props:
  sql-show: true
```

1.4.5 使用多数据源

上面的方式都是只引入一个数据源，在某些场景下，需要引入多个数据库，操作不同的数据库，此时，可以通过配置不同前缀来注入多个datasource的bean，方法如下：

- 引入依赖。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
```

```
<artifactId>spring-cloud-starter-huawei-devspore-datasource</artifactId>  
<version>${latest}</version>  
</dependency>
```

- 配置文件中配置多个数据源。

```
devspore:  
  datasource1:  
    props:  
      version: v1  
      appld: xxx  
      monitorId: xxx  
      databaseName: xxx  
      decipherClassName: xxx.xxx.xxx  
      region: az0  
    etcd:  
      address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379  
      apiVersion: v3  
      username: etcduser  
      password: etcdpwd  
      httpsEnable: false  
    sources:  
      ds1:  
        driverClassName: com.mysql.jdbc.Driver  
        jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0  
        username: datasourceuser  
        password: datasourcepwd  
        type: com.zaxxer.hikari.HikariDataSource  
      ds2:  
        driverClassName: com.mysql.jdbc.Driver  
        jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1  
        username: datasourceuser  
        password: datasourcepwd  
        type: com.zaxxer.hikari.HikariDataSource  
    router:  
      active: dc1  
      routeAlgorithm: single-read-write  
      nodes:  
        dc1:  
          master: ds1  
        dc2:  
          master: ds1  
  datasource2:  
    props:  
      version: v1  
      appld: xxx  
      monitorId: xxx  
      databaseName: xxx  
      decipherClassName: xxx.xxx.xxx  
      region: az0  
    etcd:  
      address: 127.0.0.2:2379,127.0.0.2:2379,127.0.0.2:2379  
      apiVersion: v3  
      username: etcduser  
      password: etcdpwd  
      httpsEnable: false  
    sources:  
      ds1:  
        driverClassName: com.mysql.jdbc.Driver  
        jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds0  
        username: datasourceuser  
        password: datasourcepwd  
        type: com.zaxxer.hikari.HikariDataSource  
      ds2:  
        driverClassName: com.mysql.jdbc.Driver  
        jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds1  
        username: datasourceuser  
        password: datasourcepwd  
        type: com.zaxxer.hikari.HikariDataSource  
    router:  
      active: dc1
```

```
routeAlgorithm: single-read-write
nodes:
  dc1:
    master: ds1
  dc2:
    master: ds1
```

- 根据前缀注入多个datasource。
在配置类中根据yml中的前缀，分别注入bean，并进行命名，然后即可分别操作不同的datasource。

```
@Configuration
public class App {
    @ConfigurationProperties(prefix = "devspore.datasource1")
    @Bean(name = "ds1")
    public DataSource dataSource1() {
        return new ClusterDataSource();
    }

    @ConfigurationProperties(prefix = "devspore.datasource2")
    @Bean(name = "ds2")
    public DataSource dataSource2() {
        return new ClusterDataSource();
    }
}
```

1.5 参数配置说明

1.5.1 配置项详细信息列表

MAS-DB-SDK配置分为**props**、**etcd**、**sources**、**router**四部分，在SpringBoot项目中配置项使用驼峰风格，详细信息如下表：

表 1-1 props 配置项

名称	说明	默认值	备注
props.appld	MAS应用ID	空	MAS服务应用列表页中“应用ID”列对应值。
props.monitorId	MAS监控器ID	空	MAS服务监控列表页中“监控器ID”列对应值。
props.databaseName	MAS监控DB名称	空	MAS服务监控列表页单击“编辑”中连接池配置下“数据库名”对应值。
props.decipherClassName	密码解析类	com.huawei.devspore.mas.password.DefaultDecipher	需要实现基类。 com.huawei.devspore.mas.password.Decipher
props.version	配置文件版本号	空	当前固定填v1。

名称	说明	默认值	备注
props.azs	服务所属AZ信息	空	路由算法为本地读单边写时必填。

须知

- props配置项中的值都是取自于本服务在MAS服务注册时展示的值。
- props配置项中的值与在MAS服务中的来源请参考[props、etcd配置项在MAS服务中如何查询](#)。

表 1-2 etcd 配置项

名称	说明	默认值	备注
etcd.address	ETCD地址	空	MAS服务基本信息页单击 ETCD链接地址 后的 查看详情 获取。
etcd.apiVersion	ETCD接口版本	v3	固定值v3。
etcd.username	ETCD用户名	空	MAS服务基本信息页单击 ETCD链接地址 后的 查看详情 获取。
etcd.password	ETCD密码	空	MAS服务基本信息页单击 ETCD链接地址 后的 查看详情 获取。
etcd.httpsEnable	ETCD是否启用https	false	根据ETCD实际情况选择，最新版本的MAS服务etcd均为https，此处需要配置为true，并配置证书路径。
etcd.certificatePath	启用https时证书存放路径	空	https证书存放路径。也支持不使用证书，此时不需要配置该配置项。

须知

- etcd配置项中的值都是取自于本服务在MAS服务etcd展示的值。
- props配置项中的值与在etcd中的来源请参考[props、etcd配置项在MAS服务中如何查询](#)。
- etcd.apiVersion填写固定值V3。
- etcd.httpsEnable的值取决于ETCD页面“etcd集群地址”访问是http，这里值为false；访问是https，这里值为true。查看方法参考[props、etcd配置项在MAS服务中如何查询](#)。

表 1-3 sources 数据源配置项

名称	说明	默认值	备注
sources.ds1	数据库节点名称:ds1	空	与MAS服务中连接池中的命名一致。
sources.ds1.driverClassName	数据库驱动类名	空	驱动名称，自定义。
sources.ds1.jdbcUrl	数据库连接地址	空	与MAS服务中连接池中各节点下数据库连接地址一致。
sources.ds1.userName	数据库用户名	空	与MAS服务中连接池中各节点下数据库用户名一致。
sources.ds1.password	数据库密码	空	如果密码为加密，需要自定义实现解密类配置： props.decipherClassName
sources.ds1.type	数据源类型	空	支持类型如下： <ul style="list-style-type: none">• com.zaxxer.hikari.HikariDataSource• org.apache.commons.dbcp2.BasicDataSource• com.alibaba.druid.pool.DruidDataSource

名称	说明	默认值	备注
sources.ds1.props	数据源类型其他参数	空	根据自己选择的数据源类型配置。 props: connection-timeout: 1000 validation-timeout: 1000

须知

sources主要配置本项目中涉及的一组或者多组数据源，所以上表中"ds1"为数据源的节点名称，如果有多组数据源，按照同样的格式配置多组即可。

- 对接MAS服务，各组数据源的节点名称与MAS服务中连接池中的命名一致。
- 无对接MAS服务，各组数据源的节点名称应命名为具有区别意义的名称。

表 1-4 router 路由配置

名称	说明	默认值	备注
router.active	默认数据源的激活节点	-	-
router.routeAlgorithm	路由策略	single-read-write	单边读写 single-read-write 。
router.nodes.ds1.loadBalance	读写分离负载均衡算法	空	可选项： <ul style="list-style-type: none">• RANDOM是随机。• ROUND_ROBIN是轮询。
router.nodes.ds1.master	主数据源	空	<ul style="list-style-type: none">• 当slaves为空时，读写流量都在master• 当slaves不为空时，读流量在slaves，写流量，DDL，事务操作在master。
router.nodes.ds1.slaves	读写分离读数据源列表	空	-

名称	说明	默认值	备注
router.nodes.ds1.a zs	该数据中心所属AZ 信息	空	路由算法为本地读 单边写，且未对接 MAS平台时必填。

须知

router.nodes配置罗列本项目中数据源路由节点，所以上表中“ds1”为数据源路由节点名称，如果有多组数据源，按照同样的格式配置多组即可。

- 节点router.nodes.ds1.master下配置的源名称为“表1-3”中的“sources.ds1”主节点的值。
- 节点router.nodes.ds1.slaves下配置的源名称为“表1-3”中的“sources.ds1”从节点的值。
- router.active对应值为router.nodes的一个节点名称。

1.6 不支持的接口

SDK中部分jdbc相关接口不支持，详细见SDK中com.huawei.devspore.datasource.jdbc.unsupported包。下面列出了Connection、PreparedStatement、ResultSet中的部分不支持接口，如下：

- Connection接口。

```
CallableStatement prepareCall(String sql) throws SQLException;

CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)
    throws SQLException;

CallableStatement prepareCall(
    String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws
    SQLException;

String nativeSQL(final String sql) throws SQLException;

Savepoint setSavepoint() throws SQLException;

Savepoint setSavepoint(String name) throws SQLException;

void releaseSavepoint(Savepoint savepoint) throws SQLException;

void rollback(Savepoint savepoint) throws SQLException;

void abort(Executor executor) throws SQLException;

Map<String, Class<?>> getTypeMap() throws SQLException;

void setTypeMap(Map<String, Class<?>> map) throws SQLException;

int getNetworkTimeout() throws SQLException;

void setNetworkTimeout(Executor executor, int milliseconds) throws SQLException;

Clob createClob() throws SQLException;

Blob createBlob() throws SQLException;
```

```
NClob createNClob() throws SQLException;
SQLXML createSQLXML() throws SQLException;
Struct createStruct(final String typeName, final Object[] attributes) throws SQLException;
Properties getClientInfo() throws SQLException;
String getClientInfo(String name) throws SQLException;
void setClientInfo(String name, String value);
void setClientInfo(Properties props);
void setCatalog(String catalog) throws SQLException;
void setHoldability(final int holdability) throws SQLException;
String getSchema() throws SQLException;
void setSchema(final String schema) throws SQLException;
```

- **PreparedStatement接口。**

```
void setNString(int parameterIndex, String x) throws SQLException;
void setNClob(int parameterIndex, NClob x) throws SQLException;
void setNClob(int parameterIndex, Reader x) throws SQLException;
void setNClob(int parameterIndex, Reader x, long length) throws SQLException;
void setNCharacterStream(int parameterIndex, Reader x) throws SQLException;
void setNCharacterStream(int parameterIndex, Reader x, long length);
void setRowId(int parameterIndex, RowId x) throws SQLException;
void setRef(int parameterIndex, Ref x) throws SQLException;
```

- **ResultSet接口。**

```
void updateNull(int columnIndex) throws SQLException;
void updateNull(String columnLabel) throws SQLException;
void updateBoolean(int columnIndex, boolean x) throws SQLException;
void updateBoolean(String columnLabel, boolean x) throws SQLException;
void updateByte(int columnIndex, byte x) throws SQLException;
void updateByte(String columnLabel, byte x) throws SQLException;
void updateShort(int columnIndex, short x) throws SQLException;
void updateShort(String columnLabel, short x) throws SQLException;
void updateInt(int columnIndex, int x) throws SQLException;
void updateInt(String columnLabel, int x) throws SQLException;
void updateLong(int columnIndex, long x) throws SQLException;
void updateLong(String columnLabel, long x) throws SQLException;
void updateFloat(int columnIndex, float x) throws SQLException;
void updateFloat(String columnLabel, float x) throws SQLException;
void updateDouble(int columnIndex, double x) throws SQLException;
```

```
void updateDouble(String columnLabel, double x) throws SQLException;
void updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException;
void updateBigDecimal(String columnLabel, BigDecimal x) throws SQLException;
void updateString(final int columnIndex, final String x) throws SQLException;
void updateString(String columnLabel, String x) throws SQLException;
void updateNString(int columnIndex, String nString) throws SQLException;
void updateNString(String columnLabel, String nString) throws SQLException;
void updateBytes(int columnIndex, byte[] x) throws SQLException;
void updateBytes(String columnLabel, byte[] x) throws SQLException;
void updateDate(int columnIndex, Date x) throws SQLException;
void updateDate(String columnLabel, final Date x) throws SQLException;
void updateTime(int columnIndex, Time x) throws SQLException;
void updateTime(String columnLabel, Time x) throws SQLException;
void updateTimestamp(int columnIndex, Timestamp x) throws SQLException;
void updateTimestamp(String columnLabel, Timestamp x) throws SQLException;
void updateAsciiStream(int columnIndex, InputStream inputStream) throws SQLException;
void updateAsciiStream(String columnLabel, InputStream inputStream) throws SQLException;
void updateAsciiStream(int columnIndex, InputStream x, final int length);
void updateAsciiStream(String columnLabel, InputStream x, final int length);
void updateAsciiStream(int columnIndex, InputStream inputStream, long length);
void updateAsciiStream(String columnLabel, InputStream inputStream, long length);
void updateBinaryStream(int columnIndex, InputStream x) throws SQLException ;
void updateBinaryStream(String columnLabel, InputStream x) throws SQLException;
void updateBinaryStream(int columnIndex, InputStream x, int length)
    throws SQLException;
void updateBinaryStream(String columnLabel, final InputStream x, final int length)
    throws SQLException;
void updateBinaryStream(int columnIndex, InputStream x, long length)
    throws SQLException;
void updateBinaryStream(String columnLabel, InputStream x, long length)
    throws SQLException;
void updateCharacterStream(int columnIndex, Reader x) throws SQLException;
void updateCharacterStream(String columnLabel, Reader x) throws SQLException;
void updateCharacterStream(int columnIndex, Reader x, int length)
    throws SQLException;
void updateCharacterStream(String columnLabel, Reader reader, int length);
void updateCharacterStream(int columnIndex, Reader x, long length)
    throws SQLException;
```

```
void updateCharacterStream(String columnLabel, Reader reader, long length)
    throws SQLException;

void updateNCharacterStream(int columnIndex, Reader x) throws SQLException;

void updateNCharacterStream(String columnLabel, Reader x) throws SQLException;

void updateNCharacterStream(int columnIndex, Reader x, long length)
    throws SQLException;

void updateNCharacterStream(String columnLabel, Reader x, long length)
    throws SQLException;

void updateObject(int columnIndex, Object x) throws SQLException;

void updateObject(String columnLabel, Object x) throws SQLException;

void updateObject(int columnIndex, Object x, int scaleOrLength) throws SQLException;

void updateObject(String columnLabel, Object x, int scaleOrLength)
    throws SQLException;

void updateRef(int columnIndex, Ref x) throws SQLException;

void updateRef(String columnLabel, Ref x) throws SQLException;

void updateBlob(int columnIndex, Blob x) throws SQLException;

void updateBlob(String columnLabel, Blob x) throws SQLException;

void updateBlob(int columnIndex, InputStream inputStream) throws SQLException;

void updateBlob(String columnLabel, InputStream inputStream) throws SQLException;

void updateBlob(int columnIndex, InputStream inputStream, long length)
    throws SQLException;

void updateBlob(String columnLabel, InputStream inputStream, long length)
    throws SQLException;

void updateClob(int columnIndex, Clob x) throws SQLException;

void updateClob(String columnLabel, Clob x) throws SQLException;

void updateClob(int columnIndex, Reader reader) throws SQLException;

void updateClob(String columnLabel, Reader reader) throws SQLException;

void updateClob(int columnIndex, Reader reader, long length) throws SQLException;

void updateClob(String columnLabel, Reader reader, long length) throws SQLException;

void updateNClob(int columnIndex, NClob nClob) throws SQLException;

void updateNClob(String columnLabel, NClob nClob) throws SQLException;

void updateNClob(int columnIndex, Reader reader) throws SQLException;

void updateNClob(String columnLabel, Reader reader) throws SQLException;

void updateNClob(int columnIndex, Reader reader, long length) throws SQLException;

void updateNClob(String columnLabel, Reader reader, long length) throws SQLException;

void updateArray(int columnIndex, Array x) throws SQLException;

void updateArray(String columnLabel, Array x) throws SQLException;
```

```
void updateRowId(int columnIndex, RowId x) throws SQLException;  
void updateRowId(String columnLabel, RowId x) throws SQLException;  
void updateSQLXML(int columnIndex, SQLXML xmlObject) throws SQLException;  
void updateSQLXML(String columnLabel, SQLXML xmlObject) throws SQLException;
```

1.7 常见问题

1.7.1 如何选择组件版本

- SpringBoot接入涉及组件版本参考。

表 1-5 SpringBoot 涉及组件版本

GroupId	ArtifactId	Version	备注
org.springframework.boot	spring-boot-autoconfigure-processor	2.5.4	-
org.springframework.boot	spring-boot-configuration-processor	2.5.4	-

- Spring接入涉及组件版本参考。

表 1-6 Spring 涉及组件版本

GroupId	ArtifactId	Version	备注
spring-aspects	org.springframework.aop	5.3.9	-

- 其它涉及组件版本参考。

表 1-7 其它涉及组件版本

GroupId	ArtifactId	Version	备注
io.netty	netty-codec	4.1.69.Final	-

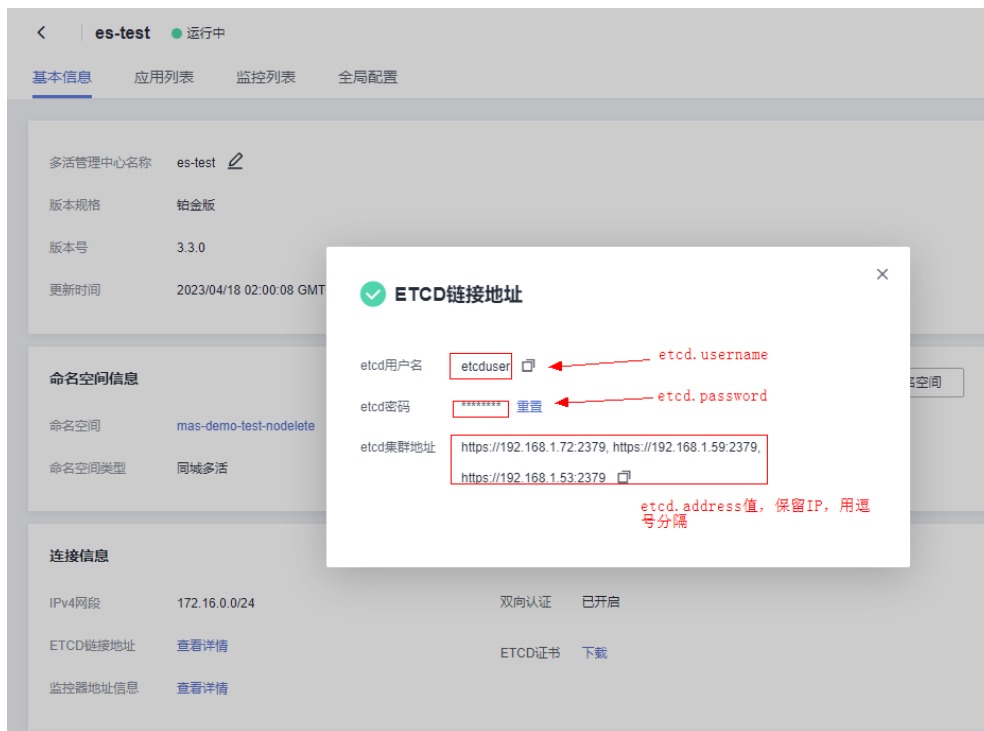
1.7.2 props、etcd 配置项在 MAS 服务中如何查询

步骤1 登录MAS控制台，进入“多活管理”页面。

步骤2 在“多活管理”页面中单击对应的实例，进入实例控制台。

步骤3 单击“应用列表”选择本项目应用，查看应用ID，即为props.appId。

- 步骤4** 单击“监控列表”选择本项目监控器，查看监控器ID，即为props.monitorId。
- 步骤5** 单击“监控列表”，在本项目监控器所在行单击“编辑”，在“编辑监控”页面单击“连接池配置”，在“数据中心”页签下的“数据源名称”就是props.databaseName。
- 步骤6** 单击“基本信息”，在“连接信息”页签单击“ETCD链接地址”右边的“查看详情”，在弹窗中查看ETCD的相关配置。



----结束

1.7.3 文档中关于 SDK 对应 yaml 配置文件的获取方式

- 步骤1** 进入MAS实例控制台“监控列表”页面。
- 步骤2** 单击“更多 > SDK接入配置”。
- 步骤3** 复制配置参数。



----结束

MAS 端配置与本地配置不一致时，哪一个生效？

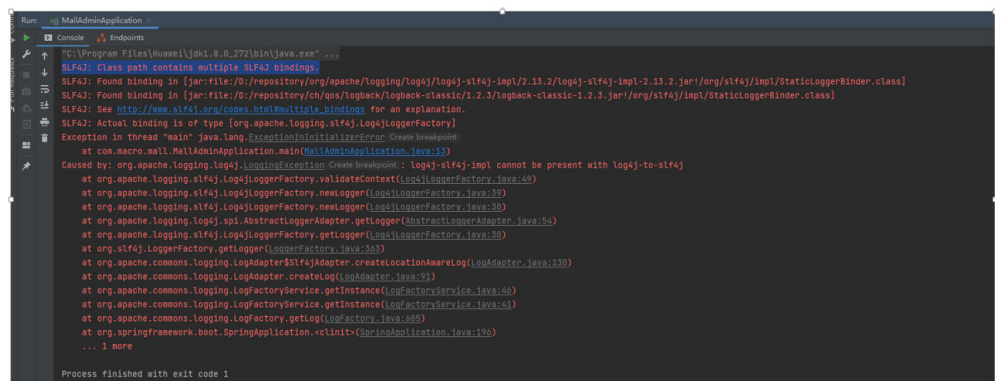
答：MAS端配置与本地配置，MAS端配置覆盖本地配置，MAS端配置生效。

1.7.4 运行 MAS-DB-SDK 问题参考

根据上述步骤配置MAS-DB-SDK后，确认相关配置信息都已包含且有效，运行项目可能会遇到一些问题，主要的问题及其相关解决过程如下：

1. Logback-classic及log4j-to-slf4j的包冲突问题。

关键错误信息如下所示：



出现原因：

对接SDK需要引入依赖包spring-cloud-starter-huawei-devspore-datasource，而本项目自带依赖包spring-boot-starter-actuator与其在Logback-classic及log4j-to-slf4j上存在冲突。

解决方案：

将对应的冲突包进行exclusion排除，排除方法如下，直接logback-classic和log4j-to-slf4j这两个包在spring-boot-starter-actuator下进行排除。排除后POM.XML显示如下：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
  <exclusions>
    <exclusion>
      <artifactId>logback-classic</artifactId>
      <groupId>ch.qos.logback</groupId>
    </exclusion>
    <exclusion>
      <artifactId>log4j-to-slf4j</artifactId>
      <groupId>org.apache.logging.log4j</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 缺少组件mybatis-spring-boot-starter。

报错信息及截图如下：

```
2022-03-21 11:22:48:697 INFO 19748 --- [main] o.s.o.w.s.t.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-03-21 11:22:48:711 INFO 19748 --- [main] o.s.o.c.StandardService : Starting service [Tomcat]
2022-03-21 11:22:48:711 INFO 19748 --- [main] o.s.o.c.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.55]
2022-03-21 11:22:48:809 INFO 19748 --- [main] o.w.s.c.ServletContextInitializer : Initializing Spring embedded WebApplicationContext
2022-03-21 11:22:48:890 INFO 19748 --- [main] o.s.w.c.context.tonger : Root WebApplicationContext: initialization completed in 4083 ms
2022-03-21 11:22:49:482 ERROR 19748 --- [main] o.s.o.w.s.t.TomcatWebServer : Error starting Tomcat context. Exception: org.springframework.beans.factory.UnsatisfiedDependencyException: Message: E
2022-03-21 11:22:49:518 INFO 19748 --- [main] o.s.o.c.StandardService : Stopping service [Tomcat]
2022-03-21 11:22:49:530 WARN 19748 --- [main] ConfigServletWebServerApplicationContext : Exception encountered during context initialization - cancelling refresh attempt: org.springframework.context.Applic
2022-03-21 11:22:49:554 INFO 19748 --- [main] o.s.b.SpringApplication : Application run failed

Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2022-03-21 11:22:49:568 ERROR 19748 --- [main] o.s.b.SpringApplication : Application run failed
org.springframework.context.ApplicationContextException: Unable to start web server; nested exception is org.springframework.boot.web.server.WebServerException: Unable to start embedded Tomcat
    at org.springframework.boot.web.servlet.context.ServletWebServerApplicationContext.onRefresh(ServletWebServerApplicationContext.java:163) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:554) ~[spring-context-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.boot.web.servlet.context.ServletWebServerApplicationContext.refresh(ServletWebServerApplicationContext.java:143) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.boot.SpringApplication.refresh(SpringApplication.java:759) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.boot.SpringApplication.refreshContext(SpringApplication.java:397) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:313) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:323) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at org.springframework.boot.SpringApplication.run(SpringApplication.java:320) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
    at com.macro.mall.MallAdminApplication.main(MallAdminApplication.java:33) [classes/:?]
Caused by: org.springframework.boot.web.server.WebServerException: Unable to start embedded Tomcat
    at org.springframework.boot.web.embedded.tomcat.TomcatWebServer.initialize(TomcatWebServer.java:103) ~[spring-boot-2.3.0.RELEASE.jar:2.3.0.RELEASE]
```

关键错误信息：

nested exception is java.lang.IllegalArgumentException: Property 'sqlSessionFactory' or 'sqlSessionTemplate' are required。

出现原因：

缺少相关依赖组件： mybatis-spring-boot-starter。

解决方案：

导入对应的依赖包，依赖包信息如下：

```
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>1.3.2</version>
</dependency>
```

3. 未匹配到正确的数据源信息。

报错信息截图如下：

```
2022-03-25 15:07:28.793 ERROR 11612 --- [main] o.s.b.w.s.t.TomcatStarter : Error starting Tomcat context. Exception: org.springframework.beans.factory.UnsatisfiedDependencyException, Message: Error
2022-03-25 15:07:28.863 INFO 11612 --- [main] o.s.a.d.c.StandardService : Stopping service [Tomcat]
2022-03-25 15:07:28.872 WARN 11612 --- [main] ConfigServletWebServerApplicationContext : Exception encountered during context initialization - cancelling refresh attempt: org.springframework.context.ApplicationCon
2022-03-25 15:07:28.890 INFO 11612 --- [main] com.taobao.tddl.datasource.logging.Listener :
Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2022-03-25 15:07:28.908 ERROR 11612 --- [main] o.s.b.d.LoggingFailureAnalysisReporter :
*****
APPLICATION FAILED TO START
*****
Description:
Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.
Reason: Failed to determine a suitable driver class
Action:
Consider the following:
If you want an embedded database (H2, HSQL or Derby), please put it on the classpath.
If you have database settings to be loaded from a particular profile you may need to activate it (no profiles are currently active).
Process finished with exit code 1
```

关键错误信息:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.报错信息显示未匹配到正确的数据源信息，程序无法启动。

出现原因:

SpringBoot自带原生数据源，相关数据源的查找是由SpringBoot自动查找并注入，但是YAML文件中将SDK提供的数据源替换SpringBoot提供的spring-dataSource后，相关依赖包无法找到数据源配置。

解决方案:

- 去除SpringBoot的spring-boot-starter-jdbc依赖包。

```
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>1.3.2</version>
  <exclusions>
    <exclusion>
      <artifactId>spring-boot-starter-jdbc</artifactId>
      <groupId>org.springframework.boot</groupId>
    </exclusion>
  </exclusions>
</dependency>
</dependencies>
```

- 配置数据SDK数据源，重启项目，将问题解决。

```
@SpringBootApplication(exclude = {JtaAutoConfiguration.class, DataSourceAutoConfiguration.class})
public class MallAdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(MallAdminApplication.class, args);
    }
}

@Primary
@ConfigurationProperties(prefix = "devspore.datasource")
@Bean(name = "ds1")
public DataSource dataSource() {
    return new ClusterDataSource();
}
```

4. 依赖包调用错误。

关键错误信息: Correct the classpath of your application so that it contains a single, compatible version of org.springframework.plugin.core.PluginRegistry, 如报错信息所示，是由于调用spring-plugin-core-2.0.0.RELEASE.jar内的一个方法时未找到，而报错。

```
2022-03-25 10:46:15.231 ERROR 8150 --- [main] o.s.b.o.LoggingFailureAnalysisReporter :
*****
APPLICATION FAILED TO START
*****
Description:
An attempt was made to call a method that does not exist. The attempt was made from the following location:
    springfox.documentation.spring.web.plugins.DocumentationPluginsManager.createContextBuilder(DocumentationPluginsManager.java:152)
The following method did not exist:
    org.springframework.plugin.core.PluginRegistry.getPluginFor(Ljava/lang/Object;Lorg/springframework/plugin/core/Plugin;)Lorg/springframework/plugin/core/Plugin;
The method's class, org.springframework.plugin.core.PluginRegistry, is available from the following locations:
    jar:file:/D:/repository/org/springframework/plugin/spring-plugin-core/2.0.0.RELEASE/spring-plugin-core-2.0.0.RELEASE.jar/org/springframework/plugin/core/PluginRegistry.class
It was loaded from the following location:
    file:/D:/repository/org/springframework/plugin/spring-plugin-core/2.0.0.RELEASE/spring-plugin-core-2.0.0.RELEASE.jar
Action:
Correct the classpath of your application so that it contains a single, compatible version of org.springframework.plugin.core.PluginRegistry
```

出现原因：

spring-plugin-core-2.0.0.RELEASE.jar对应的swagger2的版本较低。

解决方案：

当前swagger2的版本为2.9.2，要求将swagger2版本升级为3.0.0，重新加载即可。

Pom文件修改如下：

```
<swagger2.version>3.0.0</swagger2.version>
```

5. io.netty版本错误。

关键错误信息：

```
2022-03-25 16:24:33.320 ERROR 19852 --- [ault-executor-0] i.g.l.ManagedChannelImpl : [channelId= (ip://10.85.110.241:2379,100.85.126.7:2379,100.95.144.201:2379)] Uncaught exception in the Synchroniz
java.lang.NoSuchMethodError: Could not initialize class io.netty.buffer.PooledByteBufAllocator.<init>(ZIILIIIZ)V
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.access$800(Utils.java:71) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.access$800(Utils.java:71) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.NettyClientTransport.start(NettyClientTransport.java:233) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.startNewTransport(InternalSubchannel.java:238) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.access$400(InternalSubchannel.java:65) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel$2.run(InternalSubchannel.java:200) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.drain(SynchronizationContext.java:95) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.execute(SynchronizationContext.java:127) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.internal.ManagedChannelImpl$NameResolverListener.onResult(ManagedChannelImpl.java:192) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.NameResolver$Listener2.onAddresses(NameResolver.java:200) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.etcd.jetcd.resolver.IPNameResolver.doResolve(IPNameResolver.java:160) ~[jetcd-core-0.5.7.jar:?] <3 internal lines
```

并且每一段时间会跳出报错信息：

```
java.lang.NoSuchMethodError: Could not initialize class io.netty.buffer.PooledByteBufAllocator.<init>(ZIILIIIZ)V
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.access$800(Utils.java:71) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.NettyClientTransport.start(NettyClientTransport.java:233) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.startNewTransport(InternalSubchannel.java:238) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.access$400(InternalSubchannel.java:65) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel$2.run(InternalSubchannel.java:200) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.drain(SynchronizationContext.java:95) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.execute(SynchronizationContext.java:127) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.internal.ManagedChannelImpl$NameResolverListener.onResult(ManagedChannelImpl.java:192) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.NameResolver$Listener2.onAddresses(NameResolver.java:200) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.etcd.jetcd.resolver.IPNameResolver.doResolve(IPNameResolver.java:160) ~[jetcd-core-0.5.7.jar:?] <3 internal lines
```

```
2022-03-25 16:24:33.597 WARN 19852 --- [pool-5-thread-1] c.h.d.o.RemoteConfigurationWatcher : etcd error: Panic! This is a bug!
```

```
2022-03-25 16:24:33.601 ERROR 19852 --- [ault-executor-0] i.g.l.ManagedChannelImpl : [channelId= (ip://10.85.110.241:2379,100.85.126.7:2379,100.95.144.201:2379)] Uncaught exception in the Synchroniz
java.lang.NoSuchMethodError: Could not initialize class io.netty.buffer.PooledByteBufAllocator.<init>(ZIILIIIZ)V
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.access$800(Utils.java:71) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.Utils.createByteBufAllocator(Utils.java:172) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.netty.NettyClientTransport.start(NettyClientTransport.java:233) ~[grpc-netty-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.ForwardingConnectionClientTransport.start(ForwardingConnectionClientTransport.java:33) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.startNewTransport(InternalSubchannel.java:238) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel.access$400(InternalSubchannel.java:65) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.internal.InternalSubchannel$2.run(InternalSubchannel.java:200) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.drain(SynchronizationContext.java:95) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.SynchronizationContext.execute(SynchronizationContext.java:127) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.grpc.internal.ManagedChannelImpl$NameResolverListener.onResult(ManagedChannelImpl.java:192) ~[grpc-core-1.37.0.jar:1.37.0]
    at io.grpc.NameResolver$Listener2.onAddresses(NameResolver.java:200) ~[grpc-api-1.37.0.jar:1.37.0]
    at io.etcd.jetcd.resolver.IPNameResolver.doResolve(IPNameResolver.java:160) ~[jetcd-core-0.5.7.jar:?] <3 internal lines
```

```
2022-03-25 16:24:33.597 WARN 19852 --- [pool-5-thread-1] c.h.d.o.RemoteConfigurationWatcher : etcd error: Panic! This is a bug!
```

```
2022-03-25 16:24:33.601 ERROR 19852 --- [ault-executor-0] i.g.l.ManagedChannelImpl : [channelId= (ip://10.85.110.241:2379,100.85.126.7:2379,100.95.144.201:2379)] Uncaught exception in the Synchroniz
```

根因信息：

java.lang.NoSuchMethodError:io.netty.buffer.PooledByteBufAllocator.<init>(ZIILIIIZ)V。

出现原因：

SDK的使用的io.netty版本较新，而项目本身的io.netty包的版本为4.1.49相对旧，不能满足SDK的需求，故而报错。

解决方案：

对io.netty等相关的依赖包进行升级为4.1.69.Final。主要为netty-buffer，netty-common，netty-transport等包的升级，可先将原先导入的包进行排除，再重新引入较高版本的对应包。相关pom文件操作如下：

```
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-tcnative-boringssl-static</artifactId>
  <version>2.0.39.Final</version>
</dependency>
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-codec</artifactId>
  <version>4.1.69.Final</version>
  <exclusions>
    <exclusion>
      <artifactId>netty-buffer</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
    <exclusion>
      <artifactId>netty-common</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
    <exclusion>
      <artifactId>netty-transport</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-common</artifactId>
  <version>4.1.69.Final</version>
</dependency>
```

```
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-buffer</artifactId>
  <version>4.1.69.Final</version>
  <exclusions>
    <exclusion>
      <artifactId>netty-common</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-transport</artifactId>
  <version>4.1.69.Final</version>
  <exclusions>
    <exclusion>
      <artifactId>netty-buffer</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
    <exclusion>
      <artifactId>netty-common</artifactId>
      <groupId>io.netty</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

6. 容器刚启动时，etcd、mysql连接错误，后面自动恢复正常。

关键错误信息：

load data from etcd failed:

java.util.concurrent.ExecutionException:

io.etcd.jetcd.common.exception.EtcdException: io exception

at

java.util.concurrent.CompletableFuture.reportGet(CompletableFuture.java:357)

at java.util.concurrent.CompletableFuture.get(CompletableFuture.java:1915)

at

com.huawei.devspore.mas.etcd.v3impl.V3ClientImpl.get(V3ClientImpl.java:119)

at

com.huawei.devspore.datasource.config.loader.RemoteConfigurationLoader.getConfiguration(RemoteConfigurationLoader.java:102)

at

com.huawei.devspore.datasource.config.IntegrationClusterConfiguration.createIntegrationConfiguration(IntegrationClusterConfiguration.java:90)

at

com.huawei.devspore.datasource.jdbc.core.datasource.ClusterDataSource.init(ClusterDataSource.java:120)

```
at
com.huawei.devspore.datasource.jdbc.core.datasource.ClusterDataSource.init(ClusterDataSource.java:115)
at
com.huawei.devspore.datasource.jdbc.core.datasource.ClusterDataSource.<init>(ClusterDataSource.java:104)
at
com.huawei.devspore.datasource.jdbc.core.datasource.ClusterDataSourceFactory.createDataSource(ClusterDataSourceFactory.java:26)
at
com.huawei.devspore.datasource.spring.boot.ClusterSpringBootApplication.clusterDataSource(ClusterSpringBootApplication.java:60)
at
com.huawei.devspore.datasource.spring.boot.ClusterSpringBootApplication$$EnhancerBySpringCGLIB$$e947fb9e.CGLIB$clusterDataSource$2(<generated>)
at
com.huawei.devspore.datasource.spring.boot.ClusterSpringBootApplication$$EnhancerBySpringCGLIB$$e947fb9e$$FastClassBySpringCGLIB$$bfd0a96a.invoke(<generated>)
at
org.springframework.cglib.proxy.MethodProxy.invokeSuper(MethodProxy.java:228)
.....
Caused by: io.netty.channel.AbstractChannel$AnnotatedConnectException: Connection refused: /10.211.171.180:2379
Caused by: java.net.ConnectException: Connection refused
at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
at sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:717)
at
io.netty.channel.socket.nio.NioSocketChannel.doFinishConnect(NioSocketChannel.java:330)
at io.netty.channel.nio.AbstractNioChannel$AbstractNioUnsafe.finishConnect(AbstractNioChannel.java:334)
at
io.netty.channel.nio.NioEventLoop.processSelectedKey(NioEventLoop.java:707)
at
io.netty.channel.nio.NioEventLoop.processSelectedKeysOptimized(NioEventLoop.java:655)
at
io.netty.channel.nio.NioEventLoop.processSelectedKeys(NioEventLoop.java:581)
at io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:493)
at io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:986)
at io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
```

```
at
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnabl
e.java:30)
at java.lang.Thread.run(Thread.java:748)
.....
com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link
failure

The last packet sent successfully to the server was 0 milliseconds ago. The
driver has not received any packets from the server.
at
com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLEx
ceptionsMapping.java:64)
at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:836)
at com.mysql.cj.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:456)
at com.mysql.cj.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:246)
at
com.mysql.cj.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:199
)
at
com.alibaba.druid.pool.DruidAbstractDataSource.createPhysicalConnection(Dr
uidAbstractDataSource.java:1657)
at
com.alibaba.druid.pool.DruidAbstractDataSource.createPhysicalConnection(Dr
uidAbstractDataSource.java:1723)
at com.alibaba.druid.pool.DruidDataSource
$CreateConnectionThread.run(DruidDataSource.java:2801)
```

出现原因：

容器刚启动时，网络尚未正常初始化，导致连接etcd、mysql被拒绝。

解决方案：

容器启动中增加时延。

7. jackson版本冲突。

关键错误信息：

```
nested exception is java.lang.NoClassDefFoundError: com/fasterxml/jackson/
databind/PropertyNamingStrategies$KebabCaseStrategy
```

出现原因：

jackson版本冲突，低版本jackson中没有这个类。

解决方案：

排除低版本jackson依赖，jackson-core使用2.13.2，jackson-databind使用2.13.2.2，jackson-annotation使用2.13.2版本。

8. jdk版本较低，连etcd异常。

关键错误信息：

Caused by: io.netty.channel.ChannelPipelineException:
io.grpc.netty.ProtocolNegotiators\$ClientTlsHandler.handlerAdded() has thrown
an exception;

at io.grpc.netty.ProtocolNegotiators
\$ProtocolNegotiationHandler.fireProtocolNegotiationEvent(ProtocolNegotiator
s.java:1089)

... 1 more

Caused by: java.lang.RuntimeException: ALPN unsupported. Is your classpath
configured correctly? For Conscrypt, add the appropriate Conscrypt JAR

at io.grpc.netty.ProtocolNegotiators
\$ClientTlsHandler.handlerAdded0(ProtocolNegotiators.java:599)

at io.grpc.netty.ProtocolNegotiators
\$ProtocolNegotiationHandler.handlerAdder(ProtocolNegotiators.java:1048)

... 21 more

出现原因:

jdk版本较低，出问题的jdk版本为1.8.0_212，需要引入netty-tcnative-boringssl-
static依赖。

解决方案:

增加netty-tcnative-boringssl-static依赖，版本为2.0.46.Final；推荐使用
1.8.0_272以上版本jdk。

2 MAS-Redis-SDK 使用手册

- [2.1 概述](#)
- [2.2 约束](#)
- [2.3 使用场景](#)
- [2.4 接入指南](#)
- [2.5 命令参考](#)
- [2.6 参数配置说明](#)
- [2.7 敏感信息加解密](#)
- [2.8 客户各场景替换方案](#)
- [2.9 分布式锁场景最佳实践](#)

2.1 概述

2.1.1 开发简介

本文主要描述如何使用MAS-Redis-SDK在多活容灾场景下对涉及Redis的服务进行开发，结合样例讲解MAS-Redis-SDK在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Java语言，并有Java程序开发经验。
- 熟悉Maven。
- 熟悉Redis的常用操作。

MAS-Redis-SDK

MAS-Redis-SDK是一个在Jedis的基础上实现的支持多活容灾服务的Redis连接客户端，MAS-Redis-SDK的宗旨是促进开发者对于异地多活的Redis Server关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

2.1.2 开发流程

开发的流程如下所示：

1. 版本获取及引入依赖。
通过Maven引入需要的依赖，是使用MAS-Redis-SDK的基础。
2. 添加客户端配置。
通过添加客户端配置，接入MAS-Redis-SDK。
3. 创建MultiZoneClient客户端。
MAS-Redis-SDK提供读取YAML文件创建客户端的方法。
4. 按需引入客户端执行Redis操作。
在需要使用Redis客户端的地方引入MultiZoneClient，并使用MultiZoneClient执行Redis操作。

2.2 约束

2.2.1 环境约束

准备项	说明
准备操作系统	Windows系统。Windows版本要求：Windows 7及以上版本。
安装JDK	开发环境的基本配置。JDK版本要求：1.8.0_262及以上版本。
安装Maven	MAS-Redis-SDK使用Maven获取项目版本。Maven版本要求：3.3.0及以上版本。
安装和配置IntelliJ IDEA	用于开发程序的工具。IntelliJ IDEA版本要求：15.0及以上版本。

2.2.2 版本依赖

下面是MAS-Redis-SDK中引的依赖及依赖版本：

所属模块	依赖名称	依赖版本	scope
devspore-dcs	com.huaweicloud.devspore:devspore-mas-common	latest	compile
	com.fasterxml.jackson.core:jackson-core	2.15.2	compile

所属模块	依赖名称	依赖版本	scope
	com.fasterxml.jackson.core:jackson-databind	2.15.2	compile
	com.fasterxml.jackson.dataformat:jackson-dataformat-yaml	2.15.2	compile
	org.projectlombok:lombok	1.18.20	provided
	redis.clients:jedis	3.7.1	compile
	org.apache.commons:commons-pool2	2.12.0	compile
	org.yaml:snakeyaml	2.0	compile
	commons-io:commons-io	2.16.0	compile
	com.huaweicloud.devspore:devspore-mas-common	2.1.4.JDK8-RELEASE	compile
	com.google.guava:guava	32.1.2-jre	compile
	com.alibaba:transmittable-thread-local	2.14.2	compile
	org.springframework:spring-aspects	5.3.30	compile
devspore-mas-common	org.slf4j:slf4j-api	1.7.36	compile
	io.etcd:jetcd-core	0.5.7	compile
	com.google.errorprone:error_prone_annotations	2.7.1	compile
	org.projectlombok:lombok	1.18.20	provided
	com.fasterxml.jackson.core:jackson-annotations	2.15.2	compile
	io.netty:netty-codec-http2	4.1.100.Final	compile

所属模块	依赖名称	依赖版本	scope
	io.netty:netty-handler-proxy	4.1.100.Final	compile
	com.huaweicloud.sdk:huaweicloud-sdk-aom	3.0.91	compile
spring-cloud-starter-huawei-devspore-dcs	com.huaweicloud.devspore:devspore-dcs	latest	compile
	org.springframework.boot:spring-boot-autoconfigure	2.7.17	compile
	org.projectlombok:lombok	1.18.20	provided
	org.springframework.boot:spring-boot-autoconfigure-processor	2.7.17	compile
	org.springframework.boot:spring-boot-configuration-processor	2.7.17	compile
	org.springframework.data:spring-data-redis	2.7.17	compile
	org.redisson:redisson	3.23.5	compile
	com.fasterxml.jackson.datatype:jackson-datatype-jsr310	2.15.2	compile

依赖选取推荐:

依赖项	推荐版本
spring依赖	5.0.0.RELEASE及以上版本
spring-boot依赖	2.0.0.RELEASE及以上版本
org.apache.commons.commons-pool2	2.12.0及以上版本

依赖项	推荐版本
jedis客户端：redis.clients:jedis	3.7及以上版本

须知

- MAS-Redis-SDK使用的spring-boot版本为2.7.17版本，用户使用的spring-boot版本可能和该版本不一致，如果用户使用的spring-boot版本高于2.7.17版本则无需做修改，如果低于2.7.17版本需要用户将spring-boot-autoconfigure依赖修改为用户自己使用的spring-boot版本。
- 由于依赖漏洞和超期问题MAS-Redis-SDK使用的snakeyaml为2.0版本，spring-boot需要2.7.10以上版本才能使用snakeyaml2.0版本。用户可根据自身spring-boot版本确认是否使用snakeyaml2.0版本。如不使用snakeyaml2.0，需自行指定snakeyaml版本。

2.2.3 多活容灾 Redis 监控准备

MAS-Redis-SDK的多活容灾能力需要MAS服务支持，SDK本身也不支持故障动态切换数据源能力，需要配合MAS实例一起使用。

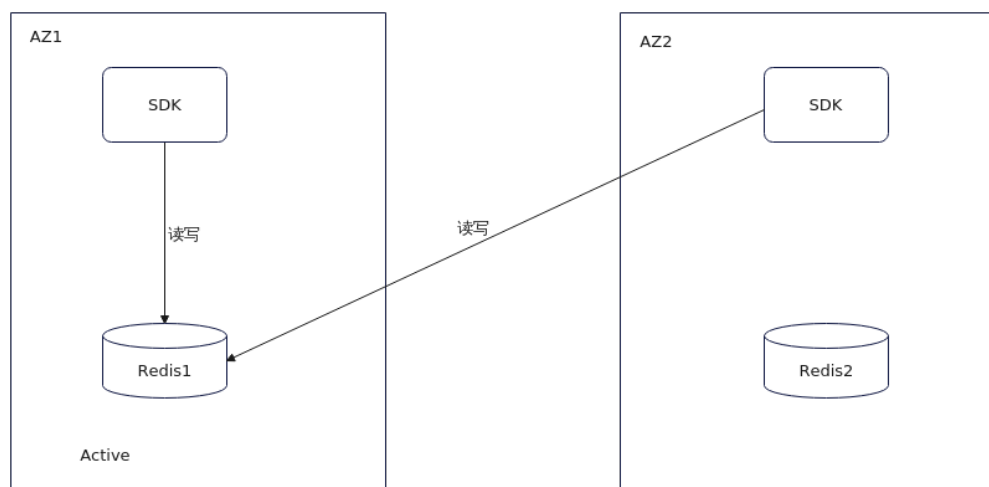
在使用MAS-Redis-SDK进行开发前，需要先做好如下准备：

1. 已创建MAS实例。
2. 在实例下已创建对应的监控器。

请参考[Redis监控管理](#)，配置多活容灾Redis监控。

2.3 使用场景

2.3.1 single-read-write (单边读写)



single-read-write场景具有以下两个特点：

- 读操作：同步路由到指定active的Redis；写操作：同步路由到指定active的Redis。
- 支持MAS动态切换激活数据源。

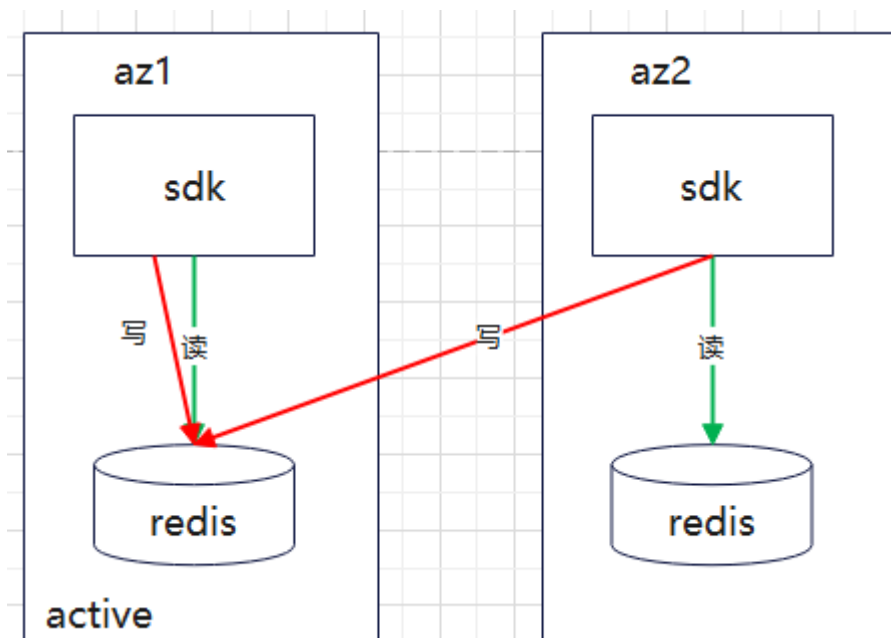
2.3.2 local-read-single-write (本地读单边写)

本地读单边写，适用于读多写少场景，读流量很大，对一致性不敏感的场景：

- 对于用户，不感知多个Redis。
- 对于SDK
 - 读操作：同步路由到local Redis。
 - 写操作：同步路由到active Redis。

此场景下Redis之间依赖同步服务相互同步。

图 2-1 local-read-single-write 部署图

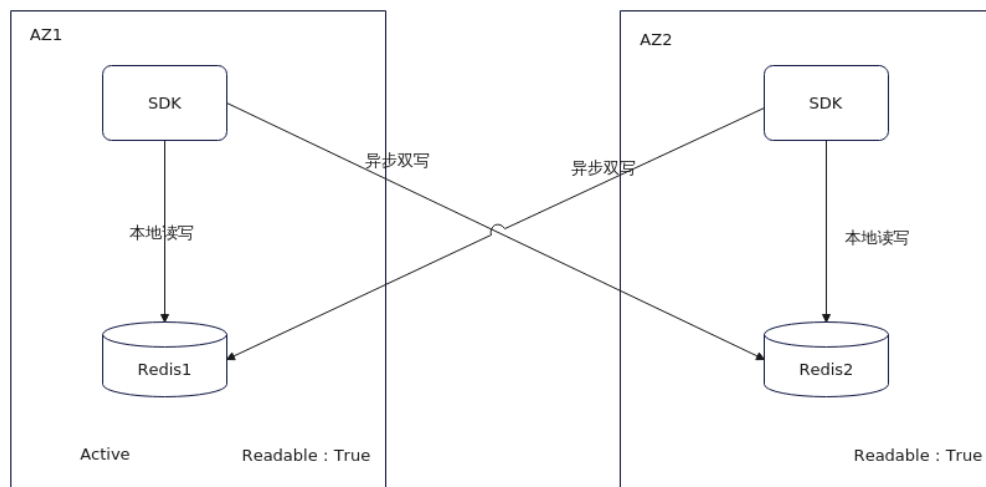


2.3.3 local-read-async-double-write (本地读异步双写)

须知

另起线程进行数据同步，不能保证异步写成功、不能保证两端数据的一致性。

图 2-2 local-read-async-double-write 部署图



读写本地异步写远端，更适用于读多写少场景，使用SDK同步两个Redis，一条写命令会先进行本地Redis的执行，成功后，异步写到远端。

本地读异步双写场景具体操作：

- 对于用户，不感知多个Redis。
- 对于SDK。
 - 读操作：同步路由到近端Redis。
 - 写操作：同步路由到近端Redis，同时异步发送到远端Redis。

此场景下Redis之间会相互同步，两个Redis位置等同没有主备之分，切换不会产生影响。

📖 说明

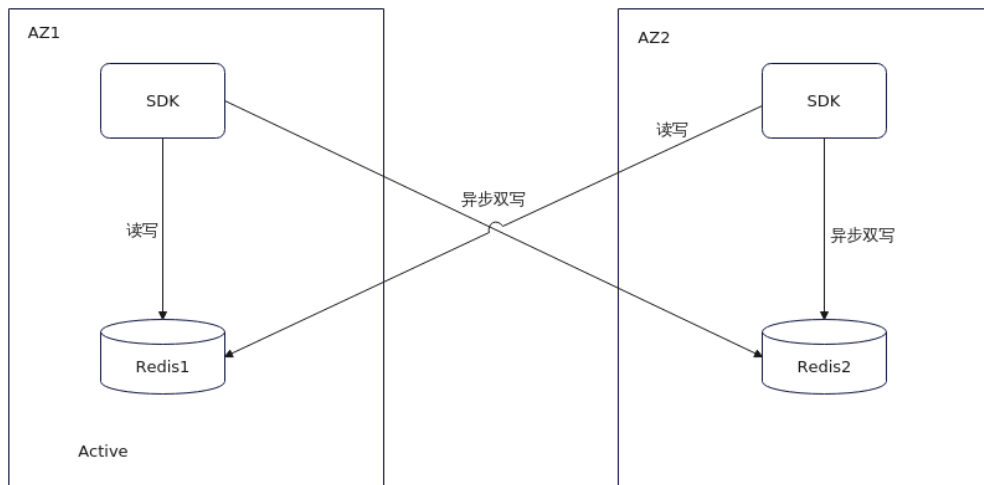
- 当不同实例内SDK操作同一个Redis key时，不能保证两个Redis的一致性。
- spop命令双写不适用。
- 双写有一定的性能影响。

2.3.4 single-read-async-double-write (单边读异步双写)

须知

另起线程进行数据同步，不能保证异步写成功、不能保证两端数据的一致性。

图 2-3 single-read-async-double-write 部署图



读写指定异步双写，适用于读多写少场景，使用SDK同步两个Redis，一条写命令会先在指定Redis上执行，成功后，异步写到另一端。

本地读异步双写场景具体操作：

- 对于用户，不感知多个Redis。
- 对于SDK。
 - 读操作：同步路由到指定的Redis。
 - 写操作：同步路由到指定的Redis，同时异步发送到远端Redis。

此场景下可切换指定的Redis。

📖 说明

- 当不同实例内SDK操作同一个Redis key时，不能保证两个Redis的一致性。
- 当SDK发生Redis切换时，切换前产生的异步双写命令和切换后主流程写操作同一个Redis的key时，两者不保证顺序：即有主流程写的值被异步双写覆盖。
- spop命令双写不适用。

2.4 接入指南

2.4.1 Spring 项目接入 MAS-Redis-SDK

步骤1 引入依赖。

📖 说明

组件版本version使用最新版本，版本的获取参考[MAS-SDK版本](#)。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>devspore-dcs</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤2 配置文件示例。

📖 说明

- 配置项详细解释参考[配置参数说明](#)。
- 根据是否接入MAS服务，确定是否配置etcd部分。
 - 接入MAS服务，etcd配置必填，以及servers部分下Redis源列表与MAS服务中配置的源保持一致。
 - 无接入MAS服务，etcd配置无需配置，servers部分下Redis源列表以实际使用为准。

```
props:
  version: v1
  app-id: xxxx
  monitor-id: xxxx
  cloud: xxxx
  region: xxxxx
  azs: az1
etcd:
  address: xxx.xxx.xxx.xxx:xxxx
  api-version: v3
  username: xxxx
  password: xxxx
  https-enable: false
redis:
  nearest: dc1
  servers:
    dc1:
      hosts: xxx.xxx.xxx.xxx:xxxx
      password: xxxxxx
      type: normal
      cloud: xxxx
      region: xxxxx
      azs: az1
      pool:
        max-total: 8
        max-idle: 8
        min-idle: 8
        max-wait-millis: 10000
        time-between-eviction-runs-millis: 60000
    dc2:
      hosts: xxx.xxx.xxx.xxx:xxxx
      password: xxxxxx
      type: normal
      cloud: xxxx
      region: xxxxx
      azs: az1
      pool:
        max-total: 8
        max-idle: 8
        min-idle: 8
        max-wait-millis: 10000
        time-between-eviction-runs-millis: 60000
route-algorithm: single-read-write
```

步骤3 MAS-Redis-SDK提供了MasRedisConfigurationLoader.load方法，可以读取YAML格式的配置，生成客户端。

```
@Bean
public MultiZoneClient createMultiZoneClient() {
    File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-cache.yaml").getFile());
    MasRedisConfiguration masRedisConfiguration = MasRedisConfigurationLoader.load(yamlFile);
    return MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);
}
```

步骤4 在需要执行Redis操作的地方引入MultiZoneClient，并使用MultiZoneClient执行Redis操作。

```
举例：
@Autowired
private MultiZoneClient client;
```

```
@Override
public void set(String key, String value) {
    client.set(key, value);
}
```

----结束

2.4.2 SpringBoot 项目接入 MAS-Redis-SDK

步骤1 MAS-Redis-SDK使用Maven获取版本，根据实际情况设置Maven远程仓库地址等相关配置。

步骤2 在pom.xml文件中引入依赖，下面mas.version版本为最新版本，依赖组件版本参考[如何选择组件版本](#)。

```
<properties>
  <!--以最新的版本号为准-->
  <mas.version>1.3.13-RELEASE</mas.version>
</properties>
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>spring-cloud-starter-huawei-devspore-dcs</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤3 参考配置文件示例devspore-cache.yaml，配置参数参考[2.6 参数配置说明](#)。

配置文件示例devspore-cache.yaml:

```
devspore:
  dcs:
    props:
      version: v1
      app-id: xxxx
      monitor-id: xxxx
      cloud: xxxx
      region: xxxx
      azs: xxxx
    etcd: # 可选
      address: xxx.xxx.xxx.xxx:xxxx
      api-version: v3
      username: xxxx
      password: xxxx
      https-enable: false
      active: dc1
    redis:
      servers:
        dc1: // 和MAS服务监控页中的名称保持一致:dc1和dc2
          hosts: xxx.xxx.xxx.xxx:xxxx
          password: xxxxxx
          type: normal
          cloud: xxxx
          region: xxxx
          azs: az1
          pool:
            max-total: 8
            max-idle: 8
            min-idle: 8
            max-wait-millis: 10000
            time-between-eviction-runs-millis: 60000
        dc2:
          hosts: xxx.xxx.xxx.xxx:xxxx
          password: xxxxxx
          type: normal
          cloud: xxxx
          region: xxxx
```

```
    azs: az1
    pool:
      max-total: 8
      max-idle: 8
      min-idle: 8
      max-wait-millis: 10000
      time-between-eviction-runs-millis: 60000
    route-algorithm: single-read-write
```

步骤4 在需要执行Redis操作的地方引入MultiZoneClient，并使用MultiZoneClient执行Redis操作。

举例：

```
@Autowired
private MultiZoneClient client;

@Override
public void set(String key, String value) {
    client.set(key, value);
}
```

----结束

2.4.3 Spring 项目接入 MAS-Redis-SDK（单实例）

📖 说明

单实例方式不对接MAS服务。

步骤1 引入依赖。

📖 说明

组件版本version使用最新版本，版本的获取参考[MAS-SDK版本](#)。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>devspore-dcs</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤2 配置文件示例。

📖 说明

配置项详细解释参考[配置参数说明](#)。

```
route-algorithm: single-read-write
active: dc1
redis:
  servers:
    dc1:
      hosts: xxx.xxx.xxx.xxx:xxxx
      password: xxxxxx
      type: normal
      pool:
        max-total: 8
        max-idle: 8
        min-idle: 8
        max-wait-millis: 10000
        time-between-eviction-runs-millis: 60000
```

步骤3 MAS-Redis-SDK提供了MasRedisConfigurationLoader.load方法，可以读取YAML格式的配置，生成客户端。

```
@Bean
public MultiZoneClient createMultiZoneClient() {
```

```
File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-cache.yaml").getFile());
MasRedisConfiguration masRedisConfiguration = MasRedisConfigurationLoader.load(yamlFile);
return MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);
}
```

步骤4 在需要执行Redis操作的地方引入MultiZoneClient，并使用MultiZoneClient执行Redis操作。

举例：

```
@Autowired
private MultiZoneClient client;

@Override
public void set(String key, String value) {
    client.set(key, value);
}
```

----结束

2.4.4 SpringBoot 项目接入 MAS-Redis-SDK（单实例）

📖 说明

单实例方式不对接MAS服务。

步骤1 引入依赖。

📖 说明

组件版本version使用最新版本，版本的获取参考[MAS-SDK版本](#)。

```
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>spring-cloud-starter-huawei-devspore-dcs</artifactId>
  <version>${version}</version>
</dependency>
```

步骤2 配置文件示例。

📖 说明

配置项详细解释参考[配置参数说明](#)。

```
devspore:
  dcs:
    route-algorithm: single-read-write
    active: dc1
    redis:
      servers:
        dc1: // 和MAS服务监控页中的名称保持一致:dc1和dc2
          hosts: xxx.xxx.xxx.xxx:xxxx
          password: xxxxxx
          type: normal
          pool:
            max-total: 20
            max-idle: 20
            min-idle: 20
            max-wait-millis: 10000
            time-between-eviction-runs-millis: 60000
```

步骤3 在需要执行Redis操作的地方引入MultiZoneClient，并使用MultiZoneClient执行Redis操作。

举例：

```
@Autowired
private MultiZoneClient client;

@Override
public void set(String key, String value) {
```

```
client.set(key, value);  
}
```

----结束

2.4.5 多客户端场景

创建多客户端实例的时候，每个客户端实例需要单独设置一个配置文件，配置文件的内容可参考Spring项目接入MAS-Redis-SDK（单实例）步骤二 配置文件示例[步骤2](#)。

如下示例创建两个客户端，配置文件名称分别为：devspore-cache1.yaml、devspore-cache2.yaml。

Spring 方式

```
@Bean  
public MultiZoneClient createMultiZoneClient1() {  
    File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-cache1.yaml").getFile());  
    MasRedisConfiguration masRedisConfiguration = MasRedisConfigurationLoader.load(yamlFile);  
    return MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);  
}  
  
@Bean  
public MultiZoneClient createMultiZoneClient2() {  
    File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-cache2.yaml").getFile());  
    MasRedisConfiguration masRedisConfiguration = MasRedisConfigurationLoader.load(yamlFile);  
    return MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);  
}
```

非 Spring 方式

```
File yamlFile1 = new File(this.getClass().getClassLoader().getResource("devspore-cache1.yaml").getFile());  
MasRedisConfiguration masRedisConfiguration1 = MasRedisConfigurationLoader.load(yamlFile);  
MultiZoneClient multiZoneClient1 = MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);  
  
File yamlFile2 = new File(this.getClass().getClassLoader().getResource("devspore-cache2.yaml").getFile());  
MasRedisConfiguration masRedisConfiguration2 = MasRedisConfigurationLoader.load(yamlFile);  
MultiZoneClient multiZoneClient2 = MultiZoneRedisFactory.createMultiZoneRedis(masRedisConfiguration);
```

2.5 命令参考

2.5.1 Redis 命令参考

MultiZoneClient集成了Redis的命令，使得在多个区域进行数据管理变得更加便捷和高效。

它提供了丰富的功能，包括数据读写、事务管理、数据类型操作等，可以满足各种复杂的业务需求。

常用命令使用方式参考如下：

```
String set(String key, String value);  
String set(byte[] key, byte[] value);  
String set(String key, String value, SetParams params);  
String set(byte[] key, byte[] value, SetParams params);  
String get(String key);  
byte[] get(byte[] key);  
Long del(String key);  
Long del(byte[] key);
```

```
Boolean exists(String key);
Boolean exists(byte[] key);
Long persist(String key);
Long persist(byte[] key);
String type(String key);
String type(byte[] key);
byte[] dump(String key);
byte[] dump(byte[] key);
String restore(String key, Long ttl, byte[] serializedValue);
String restore(byte[] key, Long ttl, byte[] serializedValue);
Long expire(String key, Long seconds);
Long expire(byte[] key, Long seconds);
Long pexpire(String key, Long seconds);
Long pexpire(byte[] key, Long milliseconds);
Long expireAt(String key, Long unixTime);
Long expireAt(byte[] key, Long unixTime);
Long pexpireAt(String key, Long millisecondsTimestamp);
Long pexpireAt(byte[] key, Long millisecondsTimestamp);
Long ttl(String key);
Long ttl(byte[] key);
Long pttl(String key);
Long pttl(byte[] key);
Long touch(String key);
Long touch(byte[] key);
Boolean setbit(String key, Long offset, boolean value);
Boolean setbit(byte[] key, long offset, boolean value);
Boolean setbit(String key, Long offset, String value);
Boolean setbit(byte[] key, Long offset, byte[] value);
Boolean getbit(String key, Long offset);
Boolean getbit(byte[] key, long offset);
Long setrange(String key, Long offset, String value);
Long setrange(byte[] key, Long offset, byte[] value);
String getrange(String key, Long startOffset, Long endOffset);
byte[] getrange(byte[] key, long startOffset, long endOffset);
String getSet(String key, String value);
byte[] getSet(byte[] key, byte[] value);
Long setnx(String key, String value);
Long setnx(byte[] key, byte[] value);
String setex(String key, Long seconds, String value);
String setex(byte[] key, Long ttl, byte[] value);
String psetex(String key, Long milliseconds, String value);
String psetex(byte[] key, Long milliseconds, byte[] value);
Long decrBy(String key, Long decrement);
Long decrBy(byte[] key, Long decrement);
Long decr(String key);
Long decr(byte[] key);
Long incrBy(String key, Long increment);
Long incrBy(byte[] key, Long increment);
Double incrByFloat(String key, Double increment);
Double incrByFloat(byte[] key, Double increment);
Long incr(String key);
Long incr(byte[] key);
Long append(String key, String value);
Long append(byte[] key, byte[] value);
String substr(String key, Integer start, Integer end);
byte[] substr(byte[] key, Integer start, Integer end);
Long strlen(String key);
Long strlen(byte[] key);
Long del(String... key);
Long del(byte[]... key);
Set<String> keys(String expression);
Set<byte[]> keys(byte[] expression);
Long unlink(final String key);
Long unlink(final byte[] key);
Long unlink(final String... keys);
Long unlink(final byte[]... keys);
ScanResult<String> scan(String cursor, ScanParams params);
ScanResult<byte[]> scan(byte[] cursor, ScanParams params);
List<String> mget(String... keys);
```

```
List<byte[]> mget(byte[]... keys);
String mset(String... keyvalues);
Long msetnx(String... keyvalues);
String rename(String oldkey, String newkey);
String rename(byte[] oldkey, byte[] newkey);
Long renamenx(String oldkey, String newkey);
Long renamenx(byte[] oldkey, byte[] newkey);
String lindex(String key, Long index);
byte[] lindex(byte[] key, Long index);
Long linsert(String key, Boolean isBefore, String pivot, String value);
Long linsert(byte[] key, Boolean isBefore, byte[] pivot, byte[] value);
Long llen(String key);
Long llen(byte[] key);
String lpop(String key);
byte[] lpop(byte[] key);
Long lpush(String key, String... string);
Long lpush(byte[] key, byte[]... string);
Long lpushx(String key, String... string);
Long lpushx(byte[] key, byte[]... string);
List<String> lrange(String key, Long start, Long stop);
List<byte[]> lrange(byte[] key, Long start, Long stop);
Long lrem(String key, Long count, String value);
Long lrem(byte[] key, Long count, byte[] value);
String lset(String key, Long index, String value);
String lset(byte[] key, Long index, byte[] value);
String ltrim(String key, Long start, Long stop);
String ltrim(byte[] key, Long start, Long stop);
String rpop(String key);
byte[] rpop(byte[] key);
List<String> rpop(String key, int count);
List<byte[]> rpop(byte[] key, int count);
String rpoplpush(String srckey, String dstkey);
byte[] rpoplpush(byte[] srckey, byte[] dstkey);
Long rpush(String key, String... string);
Long rpush(byte[] key, byte[]... string);
Long rpushx(String key, String... string);
Long rpushx(byte[] key, byte[]... string);
Long hdel(String key, String... field);
Long hdel(byte[] key, byte[]... field);
Boolean hexists(String key, String field);
Boolean hexists(byte[] key, byte[] field);
String hget(String key, String field);
byte[] hget(byte[] key, byte[] field);
Map<String, String> hgetAll(String key);
Map<byte[], byte[]> hgetAll(byte[] key);
Long hincrBy(String key, String field, Long value);
Long hincrBy(byte[] key, byte[] field, Long value);
Double hincrByFloat(String key, String field, Double value);
Double hincrByFloat(byte[] key, byte[] field, Double value);
Set<String> hkeys(String key);
Set<byte[]> hkeys(byte[] key);
Long hlen(String key);
Long hlen(byte[] key);
List<String> hmget(String key, String... fields);
List<byte[]> hmget(byte[] key, byte[]... fields);
String hmset(String key, Map<String, String> hash);
String hmset(byte[] key, Map<byte[], byte[]> hash);
Long hset(String key, String field, String value);
Long hset(byte[] key, byte[] field, byte[] value);
Long hset(String key, Map<String, String> hash);
Long hset(byte[] key, Map<byte[], byte[]> hash);
Long hsetnx(String key, String field, String value);
Long hsetnx(byte[] key, byte[] field, byte[] value);
Long hstrlen(String key, String field);
Long hstrlen(byte[] key, byte[] field);
List<String> hvals(String key);
List<byte[]> hvals(byte[] key);
ScanResult<Map.Entry<String, String>> hscan(String key, String cursor);
ScanResult<Map.Entry<byte[], byte[]>> hscan(byte[] key, byte[] cursor);
```

```
ScanResult<Map.Entry<String, String>> hscan(final String key, final String cursor, final ScanParams params);
ScanResult<Map.Entry<byte[], byte[]>> hscan(final byte[] key, final byte[] cursor, final ScanParams params);
Long sadd(final String key, final String... member);
Long sadd(final byte[] key, final byte[]... member);
Long scard(final String key);
Long scard(final byte[] key);
Set<String> sdiff(final String... keys);
Set<byte[]> sdiff(final byte[]... keys);
Long sdiffstore(final String dstkey, final String... keys);
Set<String> sinter(final String... keys);
Set<byte[]> sinter(final byte[]... keys);
Long sinterstore(final String dstkey, final String... keys);
Boolean sismember(final String key, final String member);
Boolean sismember(final byte[] key, final byte[] member);
Set<String> smembers(final String key);
Set<byte[]> smembers(final byte[] key);
Long smove(final String srckey, final String dstkey, final String member);
Long smove(final byte[] srckey, final byte[] dstkey, final byte[] member);
String spop(final String key);
byte[] spop(final byte[] key);
Set<String> spop(final String key, final Long count);
Set<byte[]> spop(final byte[] key, final Long count);
String srandmember(final String key);
byte[] srandmember(final byte[] key);
List<String> srandmember(final String key, final Integer count);
List<byte[]> srandmember(final byte[] key, final Integer count);
Long srem(final String key, final String... member);
Long srem(final byte[] key, final byte[]... member);
Set<String> sunion(final String... keys);
Set<byte[]> sunion(final byte[]... keys);
Long sunionstore(final String dstkey, final String... keys);
Long sunionstore(final byte[] dstkey, final byte[]... keys);
ScanResult<String> sscan(final String key, final String cursor);
ScanResult<byte[]> sscan(final byte[] key, final byte[] cursor);
Long zadd(String key, Double score, String member);
Long zadd(byte[] key, Double score, byte[] member);
Long zadd(String key, Double score, String member, ZAddParams params);
Long zadd(byte[] key, Double score, byte[] member, ZAddParams params);
Long zadd(String key, Map<String, Double> scoreMembers);
Long zadd(byte[] key, Map<byte[], Double> scoreMembers);
Long zadd(String key, Map<String, Double> scoreMembers, ZAddParams params);
Long zadd(byte[] key, Map<byte[], Double> scoreMembers, ZAddParams params);
Set<String> zrange(String key, Long start, Long stop);
Set<byte[]> zrange(byte[] key, Long start, Long stop);
Long zrem(String key, String... members);
Long zrem(byte[] key, byte[]... members);
Double zincrby(String key, Double increment, String member);
Double zincrby(byte[] key, Double increment, byte[] member);
Double zincrby(String key, Double increment, String member, ZIncrByParams params);
Double zincrby(byte[] key, Double increment, byte[] member, ZIncrByParams params);
Long zrank(String key, String member);
Long zrank(byte[] key, byte[] member);
Long zrevrank(String key, String member);
Long zrevrank(byte[] key, byte[] member);
Set<String> zrevrange(String key, Long start, Long stop);
Set<byte[]> zrevrange(byte[] key, Long start, Long stop);
Set<Tuple> zrangeWithScores(String key, Long start, Long stop);
Set<Tuple> zrangeWithScores(byte[] key, Long start, Long stop);
Set<Tuple> zrevrangeWithScores(String key, Long start, Long stop);
Set<Tuple> zrevrangeWithScores(byte[] key, Long start, Long stop);
Long zcard(String key);
Long zcard(byte[] key);
Double zscore(String key, String member);
Double zscore(byte[] key, byte[] member);
Tuple zpopmax(String key);
Tuple zpopmax(byte[] key);
Set<Tuple> zpopmax(String key, Integer count);
Set<Tuple> zpopmax(byte[] key, Integer count);
Tuple zpopmin(String key);
```

```
Tuple zpopmin(byte[] key);
Set<Tuple> zpopmin(String key, Integer count);
Set<Tuple> zpopmin(byte[] key, Integer count);
List<String> sort(String key);
List<byte[]> sort(byte[] key);
List<String> sort(String key, SortingParams sortingParameters);
List<byte[]> sort(byte[] key, SortingParams sortingParameters);
Long zcount(String key, Double min, Double max);
Long zcount(byte[] key, Double min, Double max);
Long zcount(String key, String min, String max);
Long zcount(byte[] key, byte[] min, byte[] max);
Set<String> zrangeByScore(String key, Double min, Double max);
Set<byte[]> zrangeByScore(byte[] key, Double min, Double max);
Set<String> zrangeByScore(String key, String min, String max);
Set<byte[]> zrangeByScore(byte[] key, byte[] min, byte[] max);
Set<String> zrevrangeByScore(String key, Double max, Double min);
Set<byte[]> zrevrangeByScore(byte[] key, Double max, Double min);
Set<String> zrangeByScore(String key, Double min, Double max, Integer offset, Integer count);
Set<byte[]> zrangeByScore(byte[] key, Double min, Double max, Integer offset, Integer count);
Set<String> zrevrangeByScore(String key, String max, String min);
Set<byte[]> zrevrangeByScore(byte[] key, byte[] max, byte[] min);
Set<String> zrangeByScore(String key, String min, String max, Integer offset, Integer count);
Set<byte[]> zrangeByScore(byte[] key, byte[] min, byte[] max, Integer offset, Integer count);
Set<String> zrevrangeByScore(String key, Double max, Double min, Integer offset, Integer count);
Set<byte[]> zrevrangeByScore(byte[] key, Double max, Double min, Integer offset, Integer count);
Set<Tuple> zrangeByScoreWithScores(String key, Double min, Double max);
Set<Tuple> zrangeByScoreWithScores(byte[] key, Double min, Double max);
Set<Tuple> zrevrangeByScoreWithScores(String key, Double max, Double min);
Set<Tuple> zrevrangeByScoreWithScores(byte[] key, Double max, Double min);
Set<Tuple> zrangeByScoreWithScores(String key, Double min, Double max, Integer offset, Integer count);
Set<Tuple> zrangeByScoreWithScores(byte[] key, Double min, Double max, Integer offset, Integer count);
Set<String> zrevrangeByScore(String key, String max, String min, Integer offset, Integer count);
Set<byte[]> zrevrangeByScore(byte[] key, byte[] max, byte[] min, Integer offset, Integer count);
Set<Tuple> zrangeByScoreWithScores(String key, String min, String max);
Set<Tuple> zrangeByScoreWithScores(byte[] key, byte[] min, byte[] max);
Set<Tuple> zrevrangeByScoreWithScores(String key, String max, String min);
Set<Tuple> zrevrangeByScoreWithScores(byte[] key, byte[] max, byte[] min);
Set<Tuple> zrangeByScoreWithScores(String key, String min, String max, Integer offset, Integer count);
Set<Tuple> zrangeByScoreWithScores(byte[] key, byte[] min, byte[] max, Integer offset, Integer count);
Set<Tuple> zrevrangeByScoreWithScores(String key, Double max, Double min, Integer offset, Integer count);
Set<Tuple> zrevrangeByScoreWithScores(byte[] key, Double max, Double min, Integer offset, Integer count);
Set<Tuple> zrevrangeByScoreWithScores(String key, String max, String min, Integer offset, Integer count);
Set<Tuple> zrevrangeByScoreWithScores(byte[] key, byte[] max, byte[] min, Integer offset, Integer count);
Long zremrangeByRank(String key, Long start, Long stop);
Long zremrangeByRank(byte[] key, Long start, Long stop);
Long zremrangeByScore(String key, Double min, Double max);
Long zremrangeByScore(byte[] key, Double min, Double max);
Long zremrangeByScore(String key, String min, String max);
Long zremrangeByScore(byte[] key, byte[] min, byte[] max);
Long zinterstore(String dstkey, String... sets);
Long zinterstore(byte[] dstkey, byte[]... sets);
Long zlexcount(String key, String min, String max);
Long zlexcount(byte[] key, byte[] min, byte[] max);
Set<String> zrangeByLex(String key, String min, String max);
Set<byte[]> zrangeByLex(byte[] key, byte[] min, byte[] max);
Set<String> zrangeByLex(String key, String min, String max, Integer offset, Integer count);
Set<byte[]> zrangeByLex(byte[] key, byte[] min, byte[] max, Integer offset, Integer count);
Set<String> zrevrangeByLex(String key, String max, String min);
Set<byte[]> zrevrangeByLex(byte[] key, byte[] max, byte[] min);
Set<String> zrevrangeByLex(String key, String max, String min, Integer offset, Integer count);
Set<byte[]> zrevrangeByLex(byte[] key, byte[] max, byte[] min, Integer offset, Integer count);
Long zremrangeByLex(String key, String min, String max);
Long zremrangeByLex(byte[] key, byte[] min, byte[] max);
Object eval(String script, Integer keyCount, String... params);
Object eval(byte[] script, Integer keyCount, byte[]... params);
Object eval(String script, List<String> keys, List<String> args);
Object eval(byte[] script, List<byte[]> keys, List<byte[]> args);
Object eval(String script, String sampleKey);
Object eval(byte[] script, byte[] sampleKey);
```

```
Object evalsha(String sha1, String sampleKey);
Object evalsha(byte[] sha1, byte[] sampleKey);
Object evalsha(String sha1, List<String> keys, List<String> args);
Object evalsha(byte[] sha1, List<byte[]> keys, List<byte[]> args);
Object evalsha(String sha1, Integer keyCount, String... params);
Object evalsha(byte[] sha1, Integer keyCount, byte[]... params);
Boolean scriptExists(String sha1, String sampleKey);
List<Boolean> scriptExists(String sampleKey, String... sha1);
String scriptLoad(String script, String sampleKey);
String scriptFlush(String sampleKey);
String scriptKill(String sampleKey);
List<Long> scriptExists(byte[]... sha1);
byte[] scriptLoad(byte[] script);
String scriptFlush();
String scriptFlush(FlushMode flushMode);
String scriptKill();
Boolean scriptExists(String sha1);
List<Boolean> scriptExists(String... sha1);
String scriptLoad(String script);
List<Long> scriptExists(byte[] sampleKey, byte[]... sha1);
byte[] scriptLoad(byte[] script, byte[] sampleKey);
String scriptFlush(byte[] sampleKey);
String scriptFlush(byte[] sampleKey, FlushMode flushMode);
String scriptKill(byte[] sampleKey);
List<String> blpop(Integer timeout, String key);
List<byte[]> blpop(Integer timeout, byte[] key);
List<String> blpop(Integer timeout, String... keys);
List<byte[]> blpop(Integer timeout, byte[]... keys);
List<String> brpop(Integer timeout, String key);
List<byte[]> brpop(Integer timeout, byte[] key);
List<String> brpop(Integer timeout, String... keys);
List<byte[]> brpop(Integer timeout, byte[]... keys);
String brpoplpush(String source, String destination, Integer timeout);
byte[] brpoplpush(byte[] source, byte[] destination, Integer timeout);
String watch(String... keys);
String watch(byte[]... keys);
String unwatch();
KeyedZSetElement bzpopmax(Double timeout, String... keys);
KeyedZSetElement bzpopmin(Double timeout, String... keys);
ScanResult<Tuple> zscan(String key, String cursor);
ScanResult<Tuple> zscan(byte[] key, byte[] cursor);
ScanResult<Tuple> zscan(final String key, final String cursor, final ScanParams params);
ScanResult<Tuple> zscan(final byte[] key, final byte[] cursor, final ScanParams params);
Long geoadd(String key, Double longitude, Double latitude, String member);
Long geoadd(byte[] key, Double longitude, Double latitude, byte[] member);
Long geoadd(String key, Map<String, GeoCoordinate> memberCoordinateMap);
Long geoadd(byte[] key, Map<byte[], GeoCoordinate> memberCoordinateMap);
List<GeoRadiusResponse> georadius(String key, Double longitude, Double latitude, Double radius, GeoUnit unit);
List<GeoRadiusResponse> georadius(byte[] key, Double longitude, Double latitude, Double radius, GeoUnit unit);
List<GeoCoordinate> geopos(String key, String... members);
List<GeoCoordinate> geopos(byte[] key, byte[]... members);
Double geodist(String key, String member1, String member2);
Double geodist(byte[] key, byte[] member1, byte[] member2);
Double geodist(String key, String member1, String member2, GeoUnit unit);
Double geodist(byte[] key, byte[] member1, byte[] member2, GeoUnit unit);
List<String> geohash(String key, String... members);
List<byte[]> geohash(byte[] key, byte[]... members);
Long publish(String channel, String message);
Long publish(byte[] channel, byte[] message);
Long pubsubNumPat();
List<String> pubsubChannels(String pattern);
List<String> pubsubChannels();
Map<String, String> pubsubNumSub(String... channels);
String psubscribe(JedisPubSub jedisPubSub, String... patterns);
String subscribe(JedisPubSub jedisPubSub, String... channels);
String psubscribe(BinaryJedisPubSub jedisPubSub, byte[]... patterns);
String subscribe(BinaryJedisPubSub jedisPubSub, byte[]... channels);
```

```
String ping();  
String info();  
String info(final String section);  
String clusterInfo();
```

2.5.2 pipeline 功能使用示例

pipeline功能使用:

- 需要返回值示例:

```
@Autowired  
private MultiZoneClient client;  
public void Demo() {  
    client.executePipeline(pipeline -> {  
        pipeline.set("devspore", "test");  
        pipeline.setnx("aaa", "bbb");  
        return pipeline.syncAndReturn();  
    });  
}
```

- 不需要返回值示例:

```
@Autowired  
private MultiZoneClient client;  
public void Demo() {  
    client.executePipeline(pipeline -> {  
        pipeline.set("devspore", "test");  
        pipeline.setnx("aaa", "bbb");  
        return null;  
    });  
}
```

2.5.3 DcsConnetion 命令参考

DcsConnection已实现部分RedisConnection的接口。

用户有需要时可以继承重写此类方法。

常用命令使用方式参考如下:

```
public Long append(@NonNull byte[] key, @NonNull byte[] value);  
public Boolean expire(@NonNull byte[] key, long seconds);  
public Boolean pExpire(@NonNull byte[] key, long millis);  
public byte[] get(@NonNull byte[] key);  
public byte[] getSet(@NonNull byte[] key, @NonNull byte[] value);  
public Boolean set(@NonNull byte[] key, @NonNull byte[] value);  
public Boolean setEx(@NonNull byte[] key, long seconds, @NonNull byte[] value);  
public Boolean setNX(@NonNull byte[] key, @NonNull byte[] value);  
public Boolean set(@NonNull byte[] key, @NonNull byte[] value, @NonNull Expiration expiration, @NonNull SetOption option);  
public void setRange(@NonNull byte[] key, @NonNull byte[] value, long offset);  
public Long exists(@NonNull byte[]... keys);  
public Long del(@NonNull byte[]... keys);  
public Long unlink(@NonNull byte[]... keys);  
public Set<byte[]> keys(@NonNull byte[] pattern);  
public Cursor<byte[]> scan(@NonNull ScanOptions options);  
public DataType type(@NonNull byte[] key);  
public Set<byte[]> sMembers(@NonNull byte[] key);  
public Long sAdd(@NonNull byte[] key, @NonNull byte[]... values);  
public Long sRem(@NonNull byte[] key, @NonNull byte[]... values);  
public Boolean sisMember(@NonNull byte[] key, @NonNull byte[] value);  
public Boolean zAdd(@NonNull byte[] key, double score, @NonNull byte[] value);  
public Long zAdd(@NonNull byte[] key, @NonNull Set<Tuple> tuples);  
public Long zAdd(@NonNull byte[] key, @NonNull Set<Tuple> set, @NonNull ZAddArgs zAddArgs);  
public Boolean zAdd(@NonNull byte[] key, double v, @NonNull byte[] member, @NonNull ZAddArgs zAddArgs);  
public Long zRem(@NonNull byte[] key, @NonNull byte[]... values);  
public Double zIncrBy(@NonNull byte[] key, double increment, @NonNull byte[] value);  
public Long zCard(@NonNull byte[] key);
```

```
public Double zScore(@NonNull byte[] key, @NonNull byte[] value);
public Long zRemRange(@NonNull byte[] key, long start, long end);
public Long zRemRangeByLex(@NonNull byte[] key, @NonNull Range range);
public Set<byte[]> zRange(@NonNull byte[] key, long start, long end);
public Set<byte[]> zRevRange(@NonNull byte[] key, long start, long end);
public Cursor<Tuple> zScan(@NonNull byte[] key, @NonNull ScanOptions options);
public Set<byte[]> zRangeByScore(@NonNull byte[] key, @NonNull Range range, @NonNull Limit limit);
public Set<byte[]> zRangeByScore(@NonNull byte[] key, @NonNull String min, @NonNull String max, long offset, long count);
public Long zRemRangeByScore(@NonNull byte[] key, Range range);
public byte[] hGet(@NonNull byte[] key, @NonNull byte[] field);
public Map<byte[], byte[]> hGetAll(@NonNull byte[] key);
public Boolean hSet(@NonNull byte[] key, @NonNull byte[] field, @NonNull byte[] value);
public Cursor<Map.Entry<byte[], byte[]>> hScan(@NonNull byte[] key, @NonNull ScanOptions options);
public void hMSet(@NonNull byte[] key, @NonNull Map<byte[], byte[]> hashes);
public Long hDel(@NonNull byte[] key, @NonNull byte[]... fields);
public Boolean hExists(@NonNull byte[] key, @NonNull byte[] field);
public Set<byte[]> hKeys(@NonNull byte[] key);
public List<byte[]> hVals(@NonNull byte[] key);
public Long hIncrBy(@NonNull byte[] key, @NonNull byte[] field, long delta);
public Double hIncrBy(@NonNull byte[] key, @NonNull byte[] field, double delta);
public List<byte[]> lRange(@NonNull byte[] key, long start, long end);
public Long rPush(@NonNull byte[] key, @NonNull byte[]... values);
public Long lInsert(@NonNull byte[] key, @NonNull Position where, @NonNull byte[] pivot, @NonNull byte[] value);
public void lSet(@NonNull byte[] key, long index, @NonNull byte[] value);
public Long lPush(@NonNull byte[] key, @NonNull byte[]... values);
public Long lLen(@NonNull byte[] key);
public byte[] lPop(@NonNull byte[] key);
public List<byte[]> blPop(int timeout, @NonNull byte[]... keys);
public byte[] rPop(@NonNull byte[] bytes);
public List<byte[]> brPop(int i, @NonNull byte[]... bytes);
public Long incr(@NonNull byte[] key);
public Long incrBy(@NonNull byte[] key, long value);
public Double incrBy(@NonNull byte[] key, double value);
public Long decr(@NonNull byte[] key);
public Long decrBy(@NonNull byte[] key, long value);
public Long ttl(@NonNull byte[] key);
public Long ttl(@NonNull byte[] key, @NonNull TimeUnit timeUnit);
public Long pTtl(@NonNull byte[] key);
public Long pTtl(@NonNull byte[] key, @NonNull TimeUnit timeUnit);
public MultiZoneClient getNativeConnection();
public boolean isSubscribed();
public Subscription getSubscription();
public void subscribe(MessageListener listener, byte[]... channels);
public Long publish(byte[] channel, byte[] message);
public void pSubscribe(MessageListener listener, byte[]... patterns);
public <T> T eval(byte[] script, ReturnType returnType, int numKeys, byte[]... keysAndArgs);
public <T> T evalSha(byte[] scriptSha, ReturnType returnType, int numKeys, byte[]... keysAndArgs);
public <T> T evalSha(String scriptSha, ReturnType returnType, int numKeys, byte[]... keysAndArgs);
public void lTrim(byte[] key, long start, long end);
public Long lRem(byte[] key, long count, byte[] value);
public Properties info();
public Properties info(String section);
public String ping();
```

2.5.4 RedisTemplate 命令与 DcsConnection 接口对应关系

RedisTemplate命令对应DcsConnection接口，请参考下表。

表 2-1 RedisTemplate 和 DcsConnection 对应关系

RedisTemplate命令	DcsConnection方法
expire(stringKey, 1L, TimeUnit.SECONDS) expire(stringKey, Duration.ofSeconds(10L))	Boolean pExpire(@NonNull byte[] key, long millis)
opsForValue().get(stringKey)	byte[] get(@NonNull byte[] key)
opsForValue().set(stringKey, stringValue)	Boolean set(@NonNull byte[] key, @NonNull byte[] value)
opsForValue().set(stringKey, stringValue, 1L, TimeUnit.SECONDS)	// Boolean setEx(@NonNull byte[] key, long seconds, @NonNull byte[] value)
opsForValue().setIfAbsent(stringKey, stringValue)	Boolean setNX(@NonNull byte[] key, @NonNull byte[] value)
opsForValue().setIfAbsent(stringKey, stringValue, 1L, TimeUnit.SECONDS)	Boolean set(@NonNull byte[] key, @NonNull byte[] value, @NonNull Expiration expiration, @NonNull RedisStringCommands.SetOption option)
opsForValue().set(stringKey, stringValue, 1L)	void setRange(@NonNull byte[] key, @NonNull byte[] value, long offset)
opsForValue().append(stringKey, stringValue)	public Long append(@NonNull byte[] key, @NonNull byte[] value)
opsForValue().decrement("numKey")	Long decr(@NonNull byte[] key)
opsForValue().decrement("numKey", 2)	Long decrBy(@NonNull byte[] key, long value)
countExistingKeys(Collections.singleton("devspore"))	Long exists(@NonNull byte[]... keys)
delete(stringKey)	Long del(@NonNull byte[]... keys)
opsForValue().increment(stringKey)	Long incr(@NonNull byte[] key)
opsForValue().increment(stringKey, 1L)	Long incrBy(@NonNull byte[] key, long value)
opsForValue().increment(stringKey, 1D)	Double incrBy(@NonNull byte[] key, double value)
opsForValue().getAndSet("appendKey", "getSetValue")	byte[] getSet(@NonNull byte[] key, @NonNull byte[] value)
keys(stringKey)	Set<byte[]> keys(@NonNull byte[] pattern)
opsForSet().members(setKey)	Set<byte[]> sMembers(@NonNull byte[] key)

RedisTemplate命令	DcsConnection方法
opsForSet().add(setKey, setValue)	Long sAdd(@NonNull byte[] key, @NonNull byte[]... values)
opsForSet().remove(setKey,setValue)	Long sRem(@NonNull byte[] key, @NonNull byte[]... values)
opsForSet().isMember(setKey, setValue)	Boolean sIsMember(@NonNull byte[] key, @NonNull byte[] value)
opsForZSet().add(zSetKey, zSetValue, 1)	Boolean zAdd(@NonNull byte[] key, double score, @NonNull byte[] value)
opsForZSet().add(zSetKey, Collections.singleton(new DefaultTypedTuple<>(zSetValue, 1d)))	Long zAdd(@NonNull byte[] key, @NonNull Set<Tuple> tuples)
opsForZSet().addIfAbsent(zSetKey, Collections.singleton(new DefaultTypedTuple<>(zSetValue, 1d)))	Long zAdd(@NonNull byte[] key, @NonNull Set<Tuple> set, @NonNull ZAddArgs zAddArgs)
opsForZSet().addIfAbsent(zSetKey, zSetValue, 1)	Boolean zAdd(@NonNull byte[] key, double v, @NonNull byte[] member, @NonNull ZAddArgs zAddArgs)
opsForZSet().zCard(zSetKey)	Long zCard(@NonNull byte[] key)
opsForZSet().rangeByScore(zSetKey, 1D, 2D)	Set<byte[]> zRangeByScore(@NonNull byte[] key, @NonNull Range range, @NonNull Limit limit)
opsForZSet().rangeByScore(zSetKey, 1D, 2D, 1L, 1L)	Set<byte[]> zRangeByScore(@NonNull byte[] key, @NonNull Range range, @NonNull Limit limit)
opsForZSet().remove(zSetKey, zSetThirdValue)	Long zRem(@NonNull byte[] key, @NonNull byte[]... values)
opsForZSet().incrementScore(zSetKey, zSetValue, 2)	Double zIncrBy(@NonNull byte[] key, double increment, @NonNull byte[] value)
opsForZSet().score(zSetKey, zSetValue)	Double zScore(@NonNull byte[] key, @NonNull byte[] value)
opsForZSet().removeRange(zSetKey, 0, 2)	Long zRemRange(@NonNull byte[] key, long start, long end)
opsForZSet().range(zSetKey, 0, 4)	Set<byte[]> zRange(@NonNull byte[] key, long start, long end)
opsForZSet().reverseRange(zSetKey, 0, 2)	Set<byte[]> zRevRange(@NonNull byte[] key, long start, long end)
opsForZSet().removeRangeByScore(zSetKey, 1, 2)	Long zRemRangeByScore(@NonNull byte[] key, Range range)

RedisTemplate命令	DcsConnection方法
Long zRemRangeByLex(@NonNull byte[] key, @NonNull Range range)	opsForZSet().removeRangeByLex(zSetKey, new RedisZSetCommands.Range().gte(zSetValue).lte(zSetSecondValue))
opsForHash().get(hKey, hValue)	byte[] hGet(@NonNull byte[] key, @NonNull byte[] field)
opsForHash().entries(hKey)	Map<byte[], byte[]> hGetAll(@NonNull byte[] key)
opsForHash().put(hKey, "field", hValue)	Boolean hSet(@NonNull byte[] key, @NonNull byte[] field, @NonNull byte[] value)
opsForHash().putAll(hKey, hashes)	hMSet(@NonNull byte[] key, @NonNull Map<byte[], byte[]> hashes)
opsForHash().delete(hKey, "field")	Long hDel(@NonNull byte[] key, @NonNull byte[]... fields)
opsForHash().hasKey(hKey, "field")	Boolean hExists(@NonNull byte[] key, @NonNull byte[] field)
opsForHash().keys(hKey)	Set<byte[]> hKeys(@NonNull byte[] key)
opsForHash().increment(hKey, "field", 1L)	Long hIncrBy(@NonNull byte[] key, @NonNull byte[] field, long delta)
opsForHash().increment(hKey, "field", 2D)	Double hIncrBy(@NonNull byte[] key, @NonNull byte[] field, double delta)
opsForHash().scan(hKey, ScanOptions.scanOptions().build())	Cursor<Map.Entry<byte[], byte[]>> hScan(@NonNull byte[] key, @NonNull ScanOptions options)
opsForHash().values(hKey)	List<byte[]> hVals(byte[] key)
opsForList().set(lKey, 0, "setValue")	void lSet(@NonNull byte[] key, long index, @NonNull byte[] value)
opsForList().range(lKey, 1, 2)	List<byte[]> lRange(@NonNull byte[] key, long start, long end)
opsForList().rightPush(lKey, lValue)	Long rPush(@NonNull byte[] key, @NonNull byte[]... values)
opsForList().leftPush(lKey, lValue)	Long lPush(@NonNull byte[] key, @NonNull byte[]... values)
opsForList().leftPop(lKey)	byte[] lPop(@NonNull byte[] key)
opsForList().leftPop(lKey, 1L, TimeUnit.SECONDS)	List<byte[]> bLPop(int timeout, @NonNull byte[]... keys)

RedisTemplate命令	DcsConnection方法
opsForList().rightPop(lKey)	byte[] rPop(@NonNull byte[] bytes)
opsForList().trim(lKey, 0, 2);	void lTrim(byte[] key, long start, long end)
opsForList().remove(lKey, 1, str2byte("setValue"))	Long lRem(byte[] key, long count, byte[] value)
opsForList().size(lKey)	Long lLen(@NonNull byte[] key)
getExpire(lKey, TimeUnit.SECONDS)	Long pTtl(@NonNull byte[] key)Long pTtl(@NonNull byte[] key, @NonNull TimeUnit timeUnit)
unlink(str2byte("unlinkKey"))	Long unlink(@NonNull byte[]... keys)
type(stringKey)	DataType type(@NonNull byte[] key)

2.6 参数配置说明

local-read-async-double-write配置示例：

```
devspore:
  dcs:
    props:
      version: v1
      app-id: xxxx
      monitor-id: xxxx
      cloud: xxxx
      region: xxxx
      azs: xxxx
    etcd: # 可选
      address: xxx.xxx.xxx.xxx:xxxx
      api-version: v3
      username: xxxx
      password: xxxx
      https-enable: false
    redis:
      nearest: dc1
      servers:
        dc1: // 和MAS服务监控页中的名称保持一致:dc1和dc2
          hosts: xxx.xxx.xxx.xxx:xxxx
          password: xxxxxx
          type: normal
          cloud: xxxx
          region: xxxx
          azs: az1
          pool:
            max-total: 8
            max-idle: 8
            min-idle: 8
            max-wait-millis: 10000
            time-between-eviction-runs-millis: 60000
        dc2:
          hosts: xxx.xxx.xxx.xxx:xxxx
          password: xxxxxx
          type: normal
          cloud: xxxx
          region: xxxx
          azs: az1
```

```
pool:
  max-total: 8
  max-idle: 8
  min-idle: 8
  max-wait-millis: 10000
  time-between-eviction-runs-millis: 60000
route-algorithm: local-read-async-double-write
```

single-read-async-double-write配置示例:

```
devspore:
  dcs:
    props:
      version: v1
      app-id: xxxx
      monitor-id: xxxx
      cloud: xxxx
      region: xxxx
      azs: xxxx
    etcd: # 可选
      address: xxx.xxx.xxx.xxx:xxxx
      api-version: v3
      username: xxxx
      password: xxxx
      https-enable: false
    active: dc1
  redis:
    servers:
      dc1: // 和MAS服务监控页中的名称保持一致:dc1和dc2
        hosts: xxx.xxx.xxx.xxx:xxxx
        password: xxxxxx
        type: normal
        cloud: xxxx
        region: xxxx
        azs: az1
        pool:
          max-total: 16
          max-idle: 16
          min-idle: 16
          max-wait-millis: 10000
          time-between-eviction-runs-millis: 60000
      dc2:
        hosts: xxx.xxx.xxx.xxx:xxxx
        password: xxxxxx
        type: normal
        cloud: xxxx
        region: xxxx
        azs: az1
        pool:
          max-total: 16
          max-idle: 16
          min-idle: 16
          max-wait-millis: 10000
          time-between-eviction-runs-millis: 60000
    route-algorithm: single-read-async-double-write
```

表 2-2 配置参数详解

参数名称	是否必选	参数类型	取值范围	描述
props	否	PropertiesConfiguration object	请参考 表2-3	MAS监控配置，配合etcd使用。同MAS-DB-SDK配置。

参数名称	是否必选	参数类型	取值范围	描述
etcd	否	EtcdConfiguration object	请参考表2-4	etcd配置，如配置，则会从远端拉取。RedisServer配置对本地配置进行覆盖。同MAS-DB-SDK配置。
redis	是	RedisClusterConfiguration object	请参考表2-5	RedisServer配置。
route-algorithm	是	String	local-read-async-double-write single-read-async-double-write single-read-write	路由算法。
active	是	String	只能是“dc1”或“dc2”	激活的Redis。

表 2-3 PropertiesConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
version	是	String	-	项目版本号。
appld	是	String	-	项目组名称。
monitorId	是	String	-	监控组名称。
cloud	是	String	-	项目部署云组。
region	是	String	-	项目部署region。
azs	是	String	-	项目部署AZ。
decipherClassName	是	String	-	自定义加密类的全类名

表 2-4 EtcdConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
address	是	String	-	Etcd地址。
apiVersion	是	String	v2/v3	Etcd版本。

参数名称	是否必选	参数类型	取值范围	描述
username	是	String	-	Etcd用户名。
password	是	String	-	Etcd密码。
httpsEnable	是	Boolean	true/false	是否启用https。
certificatePath	否	String	-	证书路径，用于https认证（>=1.2.1-RELEASE）。

表 2-5 RedisClusterConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
local	否	String	只能是“dc1”或“dc2”。	指明哪个是近端Redis。仅需要在本地读单边写模式下设置。
asyncRemoteWrite.retryTimes	是	Integer	默认为3。	异步写远端操作重试次数。
connectionPool.enable	否	Boolean	true/false，默认值为true。	是否启用连接池。
asyncRemotePool	否	ThreadPoolConfiguration object	请参考表2-6。	异步写线程池配置。
servers	是	Map of ServerConfiguration	key为dc1/dc2 单个维度请参考表2-7。	dc1, dc2的RedisServer连接配置。

表 2-6 ThreadPoolConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
threadCoreSize	否	Integer	默认为8	线程池的基本大小。
maximumPoolSize	否	Integer	默认为8	最大线程池大小。
keepAliveTime	否	Long	默认为60L	空闲线程存活时间。

参数名称	是否必选	参数类型	取值范围	描述
task-queue-size	否	Integer	默认150000	缓冲队列数。

表 2-7 RedisServerConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
hosts	是	String	-	RedisServer地址。
password	是	String	-	RedisServer密码。
type	是	String	<ul style="list-style-type: none"> • cluster • master-slave • normal 	RedisServer类型。
cloud	是	String	-	RedisServer所属云。
region	是	String	-	RedisServer所属Region。
azs	是	String	-	RedisServer所属AZ。
pool	否	ConnectionPoolConfiguration object	请参考表 2-8。	Jedis连接池配置。
timeout	否	Integer	默认为2000。	连接和执行命令的超时时间。
maxAttempts	否	Integer	默认为5。	最大重试次数。
masterName	否	String	-	type选择为sentinel哨兵模式时Redis名字。
sentinelPassword	否	String	-	哨兵模式的密码。

表 2-8 ConnectionPoolConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
maxTotal	否	int	-	最大活动对象数。
maxIdle	否	int	-	最大能够保持idle状态的 对象数。
minIdle	否	int	-	最小能够保持idle状态的 对象数。
maxWaitMillis	否	long	-	当池内没有返回对象时，最大等待时间。
lifo	否	boolean	-	设置连接对象是否后进先出，默认true。
fairness	否	boolean	-	当从池中获取资源或者将资源还回池中时 是否使用。java.util.concurrent.locks.ReentrantLock.ReentrantLock 的公平锁机制，默认为false。
minEvictableIdleTime	否	long	-	连接最小空闲时间，默认60000L。
evictorShutdownTimeout	否	long	-	驱逐线程关闭的超时间。
softMinEvictableIdleTime	否	long	-	对象空闲多久后逐出, 当空闲时间>该值且空闲连接>最大空闲数时直接逐出, 不再根据MinEvictableIdleTimeMillis判断。
numTestsPerEvictionRun	否	int	-	次释放连接的最大数目。
evictionPolicyClassName	否	String	-	设置逐出策略，默认策略为“org.apache.commons.pool2.impl.DefaultEvictionPolicy”

参数名称	是否必选	参数类型	取值范围	描述
testOnCreate	否	boolean	-	在连接对象创建时测试连接对象的有效性,默认false。
testOnBorrow	否	boolean	-	从池中获取连接时是否测试连接的有效性,默认false,建议在对性能要求不高的情况下配置为true。
testOnReturn	否	boolean	-	在连接对象返回时,是否测试对象的有效性,默认false。
testWhileIdle	否	boolean	-	在连接池空闲时是否测试连接对象的有效性,默认true(建议配置为true)。
blockWhenExhausted	否	boolean	-	当池中的资源耗尽时是否进行阻塞,设置false直接报错,true表示会一直等待,直到有可用资源。
jmxEnabled	否	boolean	-	设置是否启用JMX,默认true。
jmxNamePrefix	否	String	-	设置JMX前缀名,默认值pool。
jmxNameBase	否	String	-	设置JMX基础名。
timeBetweenEvictionRuns	否	long	-	空闲连接检测线程,检测的周期,毫秒数。如果为负值,表示不运行检测线程。默认30000L。
timeBetweenEvictionRunsMillis	否	long	-	空闲连接检测线程,检测的周期,毫秒数。如果为负值,表示不运行检测线程。默认为30000L。

2.7 敏感信息加解密

配置文件中配置密文信息。可以解密的配置有
devspore.dcs.redis.servers.dc1.password,devspore.dcs.redis.servers.dc1.sentinelPass
word,devspore.dcs.etcd.password

```
devspore:  
  dcs:  
    redis:  
      servers:  
        dc1:  
          hosts:  
            password: 密文信息  
            type: cluster
```

需做以下两步操作，实现敏感信息的解密。

步骤1 新建一个Decipher接口的实现类。

```
import com.huawei.devspore.mas.password.Decipher;  
  
public class MyDecipher implements Decipher {  
    @Override  
    public String decode(String s) {  
        if (s == null) {  
            return null;  
        }  
  
        // 使用自定义的解密算法  
        return s;  
    }  
}
```

步骤2 在配置文件中将属性devspore.dcs.props.decipherClassName配置为实现类的全类名。

```
devspore:  
  dcs:  
    props:  
      decipher-class-name: com.demo.MyDecipher
```

----结束

2.8 客户各场景替换方案

2.8.1 Jedis、Lettuce

Jedis是一个流行的Java客户端库，用于与Redis数据库进行交互。

Lettuce 是一个可伸缩线程安全的Redis客户端，多个线程可以共享同一个RedisConnection。

直接替换客户端命令参考如下。

```
import com.huawei.devspore.mas.redis.adapter.model.SetParams;  
import com.huawei.devspore.mas.redis.core.MultiZoneClient;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class DemoService{  
    @Autowired  
    private MultiZoneClient client;  
    public void set() {  
        client.set("lock", "", SetParams.builder().hasNx(true).ex(30L).build());  
    }  
}
```

```
}  
}
```

2.8.2 Spring-Boot-Data-Redis

RedisTemplate

RedisTemplate是Spring Data Redis提供的一个用于操作Redis的模板类。

它封装了对Redis的常见操作，如存储、读取、删除等，简化了与Redis交互的过程。

对接RedisTemplate命令参考如下。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;  
import com.huawei.devspore.mas.redis.spring.boot.cache.DcsConnectionFactory;  
import org.springframework.boot.autoconfigure.condition.ConditionalOnSingleCandidate;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.data.redis.core.StringRedisTemplate;  
  
@Configuration  
public class TemplateConfig {  
    @Bean  
    public DcsConnectionFactory dcsConnectionFactory(MultiZoneClient client) {  
        return new DcsConnectionFactory(client);  
    }  
  
    @Bean  
    @ConditionalOnSingleCandidate(RedisConnectionFactory.class)  
    public RedisTemplate<Object, Object> RedisTemplate(RedisConnectionFactory RedisConnectionFactory) {  
        RedisTemplate<Object, Object> template = new RedisTemplate<>();  
        template.setConnectionFactory(RedisConnectionFactory);  
        return template;  
    }  
  
    @Bean  
    @ConditionalOnSingleCandidate(RedisConnectionFactory.class)  
    public StringRedisTemplate stringRedisTemplate(RedisConnectionFactory RedisConnectionFactory) {  
        return new StringRedisTemplate(RedisConnectionFactory);  
    }  
}
```

RedisCacheManager

RedisCacheManager是一个用于管理Redis缓存的工具类。

对接RedisCacheManager命令参考如下。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;  
import com.huawei.devspore.mas.redis.spring.boot.cache.DcsConnectionFactory;  
import org.springframework.cache.CacheManager;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.cache.RedisCacheConfiguration;  
import org.springframework.data.redis.cache.RedisCacheManager;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;  
import org.springframework.data.redis.serializer.RedisSerializationContext;  
import org.springframework.data.redis.serializer.RedisSerializer;  
import org.springframework.data.redis.serializer.StringRedisSerializer;  
  
import java.time.Duration;
```

```
@Configuration
public class TemplateConfig {
    @Bean
    public DcsConnectionFactory dcsConnectionFactory(MultiZoneClient client) {
        return new DcsConnectionFactory(client);
    }

    @Bean
    public CacheManager cacheManager(RedisConnectionFactory dcsConnectionFactory) {
        RedisSerializer<String> redisSerializer = new StringRedisSerializer();
        GenericJackson2JsonRedisSerializer genericJackson2JsonRedisSerializer = new
GenericJackson2JsonRedisSerializer();

        // 配置序列化
        RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofSeconds(5)) //设置缓存失效时间
            .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(redisSerializer))
            .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(genericJackson2Jso
nRedisSerializer))
            .disableCachingNullValues();

        return RedisCacheManager.builder(dcsConnectionFactory)
            .cacheDefaults(config)
            .build();
    }
}
```

2.8.3 ShedLock

ShedLock是一个用于分布式任务调度的开源库。

它提供了一种简单而可靠的方式来确保在分布式环境中只有一个节点执行指定的任务。

对接ShedLock命令参考如下。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
import com.huawei.devspore.mas.redis.spring.boot.cache.DcsConnectionFactory;
import net.javacrumbs.shedlock.core.LockProvider;
import net.javacrumbs.shedlock.provider.redis.spring.RedisLockProvider;
import net.javacrumbs.shedlock.spring.annotation.EnableSchedulerLock;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Configuration
@EnableScheduling
@EnableSchedulerLock(defaultLockAtMostFor = "PT30S")
public class ShedLockJdbcConfig {
    @Bean
    public DcsConnectionFactory dcsConnectionFactory(MultiZoneClient client) {
        return new DcsConnectionFactory(client);
    }

    @Bean
    public LockProvider lockProvider(DcsConnectionFactory connectionFactory) {
        return new RedisLockProvider(connectionFactory);
    }
}
```

2.8.4 Redisson Lock

Redisson是一个基于Redis的Java库，它提供了一系列的分布式对象和服务，其中包括分布式锁。

Redisson的分布式锁实现了可靠的分布式锁机制，可以在分布式环境下实现对共享资源的并发访问控制。

对接Redisson Lock命令参考如下。

```
import com.huawei.devspore.mas.redis.config.Constants;
import com.huawei.devspore.mas.redis.config.MasRedisConfiguration;
import com.huawei.devspore.mas.redis.config.RedisServerConfiguration;
import com.huawei.devspore.mas.redis.config.RedisType;
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
import com.huawei.devspore.mas.redis.exception.DcsException;

import lombok.extern.slf4j.Slf4j;

import org.Redisson.Redisson;
import org.Redisson.api.RLock;
import org.Redisson.api.RedissonClient;
import org.Redisson.codec.JsonJacksonCodec;
import org.Redisson.config.Config;
import org.springframework.stereotype.Service;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

@Slf4j
@Service
public class RedissonClientStorage {
    // dcs的客户端
    private final MultiZoneClient client;

    // dc1和dc2的RedissonClient
    private final Map<String, RedissonClient> RedissonDcsMap = new HashMap<>();

    /**
     * @param client      MultiZoneClient用于获取active.
     * @param masRedisConfiguration 获取Redis配置
     * @return
     */
    public RedissonClientStorage(MultiZoneClient client, MasRedisConfiguration masRedisConfiguration) {
        this.client = client;
        if (masRedisConfiguration.getRedis().getServers().containsKey(Constants.DC_1)) {
            RedissonDcsMap.put(Constants.DC_1,
                create(masRedisConfiguration.getRedis().getServers().get(Constants.DC_1)));
        }
        if (masRedisConfiguration.getRedis().getServers().containsKey(Constants.DC_2)) {
            RedissonDcsMap.put(Constants.DC_2,
                create(masRedisConfiguration.getRedis().getServers().get(Constants.DC_2)));
        }
    }

    public static RedissonClient create(RedisServerConfiguration configuration) {
        if (RedisType.NORMAL.equals(configuration.getType())
            || RedisType.MASTER_SLAVE.equals(configuration.getType())) {
            Config config = new Config();
            config.useSingleServer()
                .setAddress(Constants.RedisSON_URI_PREFIX.concat(configuration.getHosts()))
                .setPassword(configuration.getPassword())
                .setDatabase(configuration.getDb());
            config.setCodec(new JsonJacksonCodec());
            return Redisson.create(config);
        } else if (RedisType.CLUSTER.equals(configuration.getType())) {
            Config config = new Config();
            config.useClusterServers()
                .setPassword(configuration.getPassword())
                .setNodeAddresses(Arrays.stream(configuration.getHosts().split(","))
                    .map(Constants.RedisSON_URI_PREFIX::concat)
                    .collect(Collectors.toList()));
            config.setCodec(new JsonJacksonCodec());
        }
    }
}
```

```
        return Redisson.create(config);
    } else {
        throw new DcsException(String.format("unknown redis type %s", configuration.getType()));
    }
}
/**
 * @return 活跃的RedissonClient
 */
public RedissonClient getActiveRedisson() {
    return RedissonDcsMap.get(client.getStrategyMode().getState().getActive());
}
// Redisson lock使用的demo
public void lock() throws InterruptedException {
    RedissonClient activeRedisson = this.getActiveRedisson();
    RLock lock = activeRedisson.getLock("lock");
    try {
        if (lock.tryLock()) {
            log.info("lock success-{}", Thread.currentThread());
            Thread.sleep(30000);
        } else {
            log.info("lock fail-{}", Thread.currentThread());
        }
    } finally {
        lock.unlock();
        log.info("unlock success-{}", Thread.currentThread());
    }
}
}
```

2.8.5 自定义 DcsConnection

使用场景：使用RedisTemplate命令时，DcsConnection中的接口没有覆盖到的情况，可以自定义扩展DcsConnection。

1. 自定义DcsConnection。

a. type选择为normal时使用。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
import com.huawei.devspore.mas.redis.spring.boot.cache.DcsConnection;

public class CusConnection extends DcsConnection {
    public CusConnection(MultiZoneClient client) {
        super(client);
    }
    // template中不支持但dcs中有的命令，可通过重写相应方法实现
    @Override
    public Long hLen(byte[] key) {
        return client.hlen(key);
    }
}
```

b. type选择为cluster时使用。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
import com.huawei.devspore.mas.redis.spring.boot.cache.DcsClusterConnection;

public class CusClusterConnection extends DcsClusterConnection {
    public CusClusterConnection(MultiZoneClient client) {
        super(client);
    }
    // template中不支持但dcs中有的命令，可通过重写相应方法实现
    @Override
    public Long hLen(byte[] key) {
        return client.hlen(key);
    }
}
```

2. 自定义RedisConnectionFactory。

```
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
```

```
import org.springframework.dao.DataAccessException;
import org.springframework.data.redis.connection.RedisClusterConnection;
import org.springframework.data.redis.connection.RedisConnection;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisSentinelConnection;

public class CusConnectionFactory implements RedisConnectionFactory {
    private final MultiZoneClient client;

    public CusConnectionFactory(MultiZoneClient client) {
        this.client = client;
    }
    @Override
    public RedisConnection getConnection() {
        // 单机或主从模式使用
        return new CusConnection(client);
    }
    @Override
    public RedisClusterConnection getClusterConnection() {
        // 集群模式使用
        return new CusClusterConnection(client);
    }
    @Override
    public boolean getConvertPipelineAndTxResults() {
        return false;
    }
    @Override
    public RedisSentinelConnection getSentinelConnection() {
        return null;
    }
    @Override
    public DataAccessException translateExceptionIfPossible(RuntimeException e) {
        return null;
    }
}
```

3. 配置类中指定自定义的RedisConnectionFactory。

```
@Configuration
public class TemplateConfig {
    @Bean
    public CusConnectionFactory dcsConnectionFactory(MultiZoneClient client) {
        return new CusConnectionFactory(client);
    }

    @Bean
    @Primary
    @ConditionalOnSingleCandidate(CusConnectionFactory.class)
    public RedisTemplate<Object, Object> RedisTemplate(CusConnectionFactory
RedisConnectionFactory) {
        RedisTemplate<Object, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(RedisConnectionFactory);
        return template;
    }
}
```

2.9 分布式锁场景最佳实践

分布式锁场景需根据所选取的路由模式来选择合适的策略。

1. 路由模式为single-read-write, single-read-async-double-write时
由于读写都在同一边，分布式锁不受影响。
2. 路由模式为local-read-single-write, local-read-async-double-write时
由于读写可能不在同一侧，导致分布式锁锁不住。要实现分布式锁必须保证读写在同一侧。

3. 通过setnx等命令实现的情况，需在方法上加注解路由指定读数据源@ReadRoute(from = RedisSource.ACTIVE)。
4. 如使用Redisson分布式锁时需根据激活的数据源创建RedissonClient。

```
import com.huawei.devspore.mas.redis.config.Constants;
import com.huawei.devspore.mas.redis.config.MasRedisConfiguration;
import com.huawei.devspore.mas.redis.config.RedisServerConfiguration;
import com.huawei.devspore.mas.redis.config.RedisType;
import com.huawei.devspore.mas.redis.core.MultiZoneClient;
import com.huawei.devspore.mas.redis.exception.DcsException;

import lombok.extern.slf4j.Slf4j;

import org.Redisson.Redisson;
import org.Redisson.api.RLock;
import org.Redisson.api.RedissonClient;
import org.Redisson.codec.JsonJacksonCodec;
import org.Redisson.config.Config;
import org.springframework.stereotype.Service;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

@Slf4j
@Service
public class RedissonClientStorage {
    // dcs的客户端
    private final MultiZoneClient client;

    // dc1和dc2的RedissonClient
    private final Map<String, RedissonClient> RedissonDcsMap = new HashMap<>();

    /**
     * @param client      MultiZoneClient用于获取active.
     * @param masRedisConfiguration 获取Redis配置
     * @return
     */
    public RedissonClientStorage(MultiZoneClient client, MasRedisConfiguration masRedisConfiguration) {
        this.client = client;
        if (masRedisConfiguration.getRedis().getServers().containsKey(Constants.DC_1)) {
            RedissonDcsMap.put(Constants.DC_1,
                create(masRedisConfiguration.getRedis().getServers().get(Constants.DC_1)));
        }
        if (masRedisConfiguration.getRedis().getServers().containsKey(Constants.DC_2)) {
            RedissonDcsMap.put(Constants.DC_2,
                create(masRedisConfiguration.getRedis().getServers().get(Constants.DC_2)));
        }
    }

    public static RedissonClient create(RedisServerConfiguration configuration) {
        if (RedisType.NORMAL.equals(configuration.getType())
            || RedisType.MASTER_SLAVE.equals(configuration.getType())) {
            Config config = new Config();
            config.useSingleServer()
                .setAddress(Constants.RedisSON_URI_PREFIX.concat(configuration.getHosts()))
                .setPassword(configuration.getPassword())
                .setDatabase(configuration.getDb());
            config.setCodec(new JsonJacksonCodec());
            return Redisson.create(config);
        } else if (RedisType.CLUSTER.equals(configuration.getType())) {
            Config config = new Config();
            config.useClusterServers()
                .setPassword(configuration.getPassword())
                .setNodeAddresses(Arrays.stream(configuration.getHosts().split(","))
                    .map(Constants.RedisSON_URI_PREFIX::concat)
                    .collect(Collectors.toList()));
            config.setCodec(new JsonJacksonCodec());
            return Redisson.create(config);
        }
    }
}
```

```
    } else {
        throw new DcsException(String.format("unknown redis type %s", configuration.getType()));
    }
}
/**
 * @return active的RedissonClient
 */
public RedissonClient getActiveRedisson() {
    return RedissonDcsMap.get(client.getStrategyMode().getState().getActive());
}
// Redisson lock使用的demo
public void lock() throws InterruptedException {
    RedissonClient activeRedisson = this.getActiveRedisson();
    RLock lock = activeRedisson.getLock("lock");
    try {
        if (lock.tryLock()) {
            log.info("lock success-{}", Thread.currentThread());
            Thread.sleep(30000);
        } else {
            log.info("lock fail-{}", Thread.currentThread());
        }
    } finally {
        lock.unlock();
        log.info("unlock success-{}", Thread.currentThread());
    }
}
}
```

3 MAS-ElasticSearch-SDK 使用手册

- [3.1 概述](#)
- [3.2 多活容灾ElasticSearch监控准备](#)
- [3.3 使用场景](#)
- [3.4 接入指南](#)
- [3.5 SingleReadWriteClient命令参考](#)
- [3.6 参数配置说明](#)

3.1 概述

本文主要描述如何使用MAS-ElasticSearch-SDK在多活容灾场景下对涉及ElasticSearch的服务进行开发，结合样例讲解MAS-ElasticSearch-SDK在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Java语言，并有Java程序开发经验。
- 熟悉Maven。
- 熟悉ElasticSearch的常用操作。

3.2 多活容灾 ElasticSearch 监控准备

MAS-ElasticSearch-SDK的多活容灾能力需要MAS服务支持，SDK的故障动态切换数据源能力需要配合MAS实例一起使用。

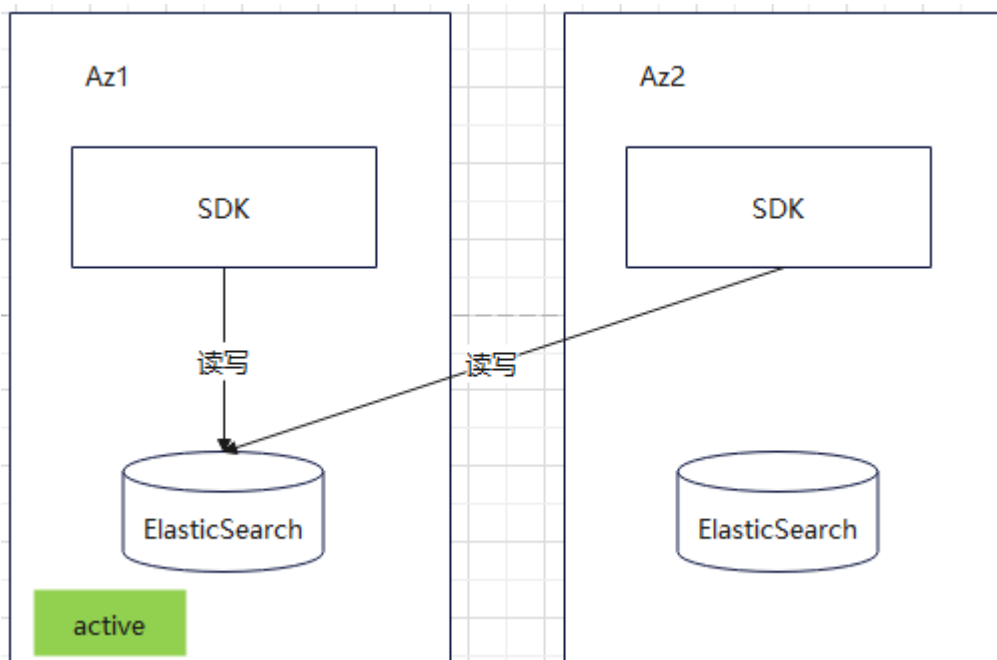
在使用MAS-ElasticSearch-SDK进行开发前，需要先做好如下准备：

1. 已创建MAS实例。
2. 在实例下已创建对应的监控器。

请参考[ElasticSearch监控管理](#)，配置多活容灾ElasticSearch监控。

3.3 使用场景

3.3.1 local-read-write (单边读写)



single-read-write场景具有以下两个特点：

1. 读操作：同步路由到指定active的elasticsearch；写操作：同步路由到指定active的elasticsearch。
2. 支持MAS动态切换激活数据源。

3.4 接入指南

3.4.1 SpringBoot 项目接入 MAS-ElasticSearch-SDK

步骤1 MAS-ElasticSearch-SDK使用Maven获取版本，根据实际情况设置Maven远程仓库地址等相关配置。

步骤2 在pom.xml文件中引入依赖。

```
<properties>
  <!--以最新的版本号为准-->
  <mas.version>1.2.6-RELEASE</mas.version>
</properties>
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>devspore-css</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤3 配置文件。

```
devspore:
  css:
    route-algorithm: single-read-write
    active: dc1
  es:
    servers:
      dc1:
        hosts: https://xxx:9200
```

```
username: admin
password: xxx
supportXPack: true
skipCertificateValid: true
certificatePath: xxx
certificatePassword: xxx
```

步骤4 引入bean, SingleReadWriteClient进行elasticsearch相关操作。

----结束

Spring Data ElasticSearch 接入

接入流程同上。

3.5 SingleReadWriteClient 命令参考

读命令: `public<T> T executeRead(Function<RestHighLevelClient, T> command)`

写命令: `public<T> T executeWrite(Function<RestHighLevelClient, T> command)`

其中RestHighLevelClient为Elasticsearch官方客户端。

3.6 参数配置说明

全部配置

参数名称	是否必选	参数类型	取值范围	描述
devspore.css.etcd	否	参考MAS-DB-SDK	-	-
devspore.css.prop	否	参考MAS-DB-SDK	-	-
devspore.css.esservers	是	Map<String, ElasticSearchServerConfiguration>	-	es配置信息。
devspore.css.active	是	String	dc1/dc2	处于active的es。
devspore.css.route-algorithm	是	String	local-read-write	路由算法。

ElasticSearchServerConfiguration

参数名称	是否必选	参数类型	取值范围	描述
hosts	是	String	-	es地址： <ip:port>或 <http:// ip:port>多个用 “;” 分隔。
username	否	String	-	es用户名。
password	否	String	-	es密码。
protocol	否	String	http/https	协议。
supportXPack	否	Boolean	默认false	是否支持 XPack验证。
skipCertificate Valid	否	Boolean	默认false	是否跳过证书 验证。
certificatePat h	否	String	-	证书地址。
certificatePass word	否	String	-	证书密码。

4 MAS-Mongo-SDK 使用手册

- [4.1 概述](#)
- [4.2 约束](#)
- [4.3 接入指南](#)
- [4.4 使用场景](#)
- [4.5 配置参数说明](#)
- [4.6 常见问题](#)

4.1 概述

4.1.1 开发简介

本文主要描述如何使用MAS-Mongo-SDK在多活容灾场景下对涉及MongoDB的服务进行开发，结合样例讲解MAS-Mongo-SDK在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Java语言，并有Java程序开发经验。
- 熟悉Maven。
- 熟悉Mongo的常用操作。

MAS-Mongo-SDK

MAS-Mongo-SDK是一个在mongo-java-driver的基础上实现的支持多活容灾服务的MongoDB连接客户端，支持MongoClient和MongoTemplate两种使用方式。MAS-Mongo-SDK的宗旨是促进开发者对于异地多活的MongoDB Server关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

4.1.2 开发流程

开发的流程如下所示：

1. 版本获取及引入依赖

- 通过Maven引入需要的依赖，是使用MAS-Mongo-SDK的基础。
2. 添加客户端配置
通过添加客户端配置，接入MAS-Mongo-SDK。
 3. 创建MongoClient客户端
MAS-Mongo-SDK提供读取YAML文件创建客户端的方法。
 4. 按需引入客户端执行MongoClient操作
在需要使用MongoDB客户端的地方引入MongoClient，并使用MongoClient执行Mongo操作。

4.2 约束

MAS-Mongo-SDK的多活容灾能力需要数据同步服务和MAS DCG服务支持，SDK本身不支持数据同步，SDK本身也不支持故障动态切换数据源能力，需要配合MAS DCG服务一起使用。

4.2.1 版本约束

MAS-Mongo-SDK对使用到的工具及相关组件的版本有所约束，如下表：

约束项	约束版本
JDK	1.8.0_262及以上版本
Maven	3.3.0及以上版本
MongoDB版本	4.4及以下版本
SDK组件依赖	见 如何选择组件版本
spring-boot版本	2.4.0及以上版本

4.2.2 多活容灾 MongoDB 监控准备

在使用MAS-MongoDB-SDK进行开发前，需要先做好如下准备：

1. 已创建MAS实例。
2. 在实例下已创建对应的监控器。

配置多活容灾MongoDB监控，请参考[MongoDB监控管理](#)。

4.3 接入指南

登录多活高可用服务控制台>帮助中心>SDK下载>MongoDB, 获取SDK jar包。

4.3.1 Spring 项目接入 MAS-Mongo-SDK

步骤1 MAS-Mongo-SDK使用Maven获取相关依赖，根据实际情况设置Maven远程仓库地址等相关配置。

步骤2 在pom.xml文件中引入如下依赖，下面mas.version版本为最新版本，依赖组件版本参考[如何选择组件版本](#)。

```
<properties>
  <!--以最新的版本号为准-->
  <mas.version>1.3.0-RELEASE</mas.version>
</properties>
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>devspore-dds</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤3 参照配置文件示例，结合[配置参数说明](#)，按实际情况修改配置文件。

配置文件示例devspore-mongo.yaml:

```
props:
  version: v1
  app-id: xxxx #MAS应用id
  monitor-id: xxxx #MAS监控id
  dbName: xxxx #MAS监控的Mongo数据库
  cloud: xxxx
  region: xxxxx
  azs: az1
etcd:
  address: 127.0.0.1:2379 #格式为ip:port, 集群模式时多个地址以英文逗号`,`分隔
  api-version: v3
  username: xxxx
  password: xxxx
  https-enable: true
  certificatePath: xxx
sources:
  dc1:
    username: xxx
    password: xxx
    url: mongodb://127.0.0.1:27017/mongo #此url请勿配置username和password
  dc2:
    username: yyy
    password: yyy
    url: mongodb://127.0.0.1:27017/mongo
active: dc1
```

步骤4 MAS-Mongo-SDK提供了MasClusterConfigurationLoader.load方法，可以读取YAML格式的配置文件，生成Cluster (MongoClient) 或MongoTemplate进行MongoDB操作。

代码示例:

```
@Bean
public Cluster createCluster() {
  File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-mongo.yaml").getFile());
  ClusterConfiguration config = MasClusterConfigurationLoader.load(yamlFile);
  return MongoClientFactory.createMongoClient(config);
}

@Bean
public MongoTemplate createMongoTemplate() {
  File yamlFile = new File(this.getClass().getClassLoader().getResource("devspore-mongo.yaml").getFile());
  ClusterConfiguration config = MasClusterConfigurationLoader.load(yamlFile);
  return MongoTemplateFactory.createMongoTemplate(config);
}
```

步骤5 在需要执行MongoDB操作的地方引入Cluster或MongoTemplate，并使用其执行Mongo操作。

----结束

4.3.2 SpringBoot 项目接入 MAS-Mongo-SDK

步骤1 MAS-Mongo-SDK使用Maven获取相关，根据实际情况设置Maven远程仓库地址等相关配置。

步骤2 在pom.xml文件中引入以下依赖，下面mas.version版本为最新版本，依赖组件版本参考[如何选择组件版本](#)。

```
<properties>
  <!--以最新的版本号为准-->
  <mas.version>1.3.0-RELEASE</mas.version>
</properties>
<dependency>
  <groupId>com.huaweicloud.devspore</groupId>
  <artifactId>spring-cloud-starter-huawei-devspore-dds</artifactId>
  <version>${mas.version}</version>
</dependency>
```

步骤3 参考配置文件示例application.yaml，配置参数参考[配置参数说明](#)。

在项目的application.yaml中添加以下配置项：

```
devspore:
  dds:
    # 基础信息 - 可选，当配置etcd后必选
    props:
      version: v1 #项目版本号，自定义
      app-id: xxxx #应用id，从MAS服务实例页面查询获取
      monitor-id: xxxx #监控器id，从MAS服务实例页面查询获取MAS监控id
      databaseName: xxxx #MAS监控的Mongo数据库
      cloud: xxxx
      region: xxxx
      azs: az1
    # etcd配置，对接MAS服务关键配置，如不对接MAS则无需配置
    etcd:
      address: 127.0.0.1:2379 #etcd地址，从MAS服务实例页面查询获取
      api-version: v3 #etcd版本，v3
      username: etcduser #etcd用户名，从MAS服务实例页面查询获取
      password: etcdpwd #etcd密码，从MAS服务实例页面查询获取
      https-enable: true #是否启用https，默认为false
      certificatePath: xxx #当启用https时证书路径，实例未开启双向认证或者不启用https场景，此项可不填
    # mongo数据源配置 - 必选
    sources:
      dc1:
        username: dc1user #mongo用户名
        password: dc2pwd #mongo密码
        url: mongodb://127.0.0.1:29017/mongo1 #mongo连接串(注意不能带用户名密码)
        azs: az1 #此mongo数据库所属AZ
      dc2:
        username: dc2user #mongo用户名
        password: dc2pwd #mongo密码
        url: mongodb://127.0.0.1:29018/mongo1 #mongo连接串(注意不能带用户名密码)
        azs: az2 #此mongo数据库所属AZ
    active: dc1
    routeStrategy: local-read-single-write #路由策略
```

步骤4 在需要执行Mongo操作的地方引入Cluster（替代原有的MongoClient）或者MongoTemplate，执行相关的Mongo操作。

举例：

```
@Autowired
private Cluster cluster;

@Autowired
MongoTemplate mongoTemplate;
```

----结束

4.4 使用场景

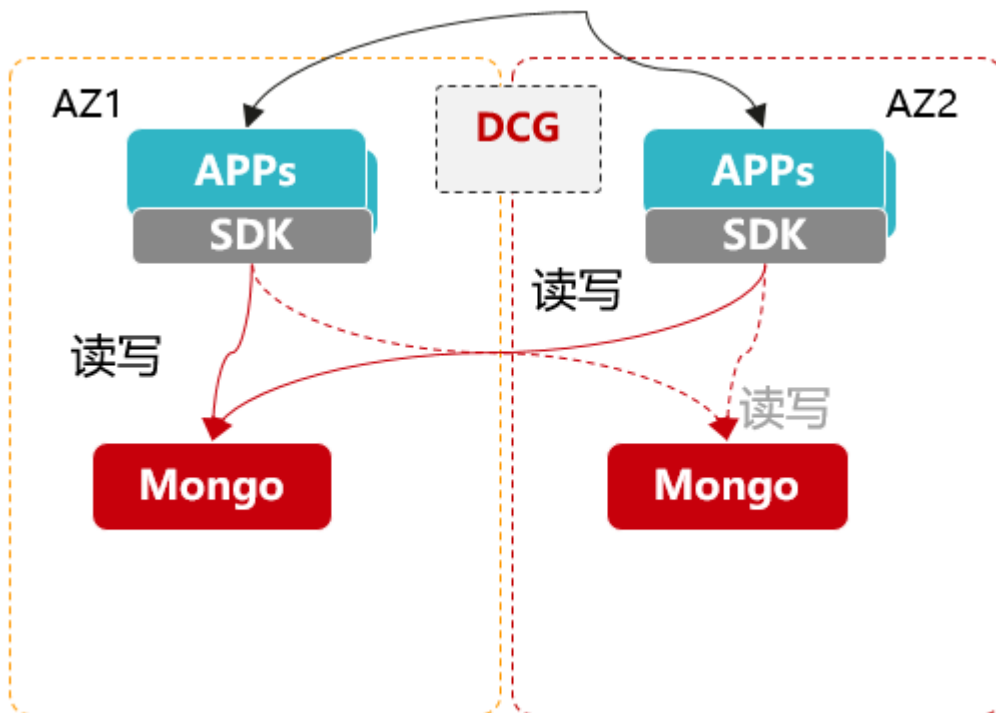
MAS-Mongo-SDK提供多活容灾能力。

多活容灾能力是指在同城场景下实现多活故障自动切换，由SDK和MAS DCG服务配合完成。

MAS-Mongo-SDK支持单边读写和本地读单边写两种模式，默认为单边读写模式。

4.4.1 单边读写

此模式为MAS-Mongo-SDK的默认路由模式，在单边读写模式下，对Mongo的读写操作都在同一Mongo数据库(dc1或dc2)进行，通过配置文件中的active字段或MAS界面配置活跃节点来指定。（通过注解指定数据源的场景例外，注解强制指定路由见[4.4.3 强制路由](#)）。



配置示例：

```
devspore:
  dds:
    # 基础信息 - 可选，当配置etcd后必选
    props:
      version: v1 #项目版本号，自定义
      app-id: xxxx #应用id，从MAS服务实例页面查询获取
      monitor-id: xxxx #监控器id，从MAS服务实例页面查询获取MAS监控id
      databaseName: xxxx #MAS监控的Mongo数据库
    azs: az1
    # etcd配置，对接MAS服务关键配置，如不对接MAS则无需配置
    etcd:
      address: 127.0.0.1:2379 #etcd地址，从MAS服务实例页面查询获取
      api-version: v3 #etcd版本，v3
      username: etcduser #etcd用户名，从MAS服务实例页面查询获取
      password: etcdpwd #etcd密码，从MAS服务实例页面查询获取
      https-enable: true #是否启用https，默认为false
```

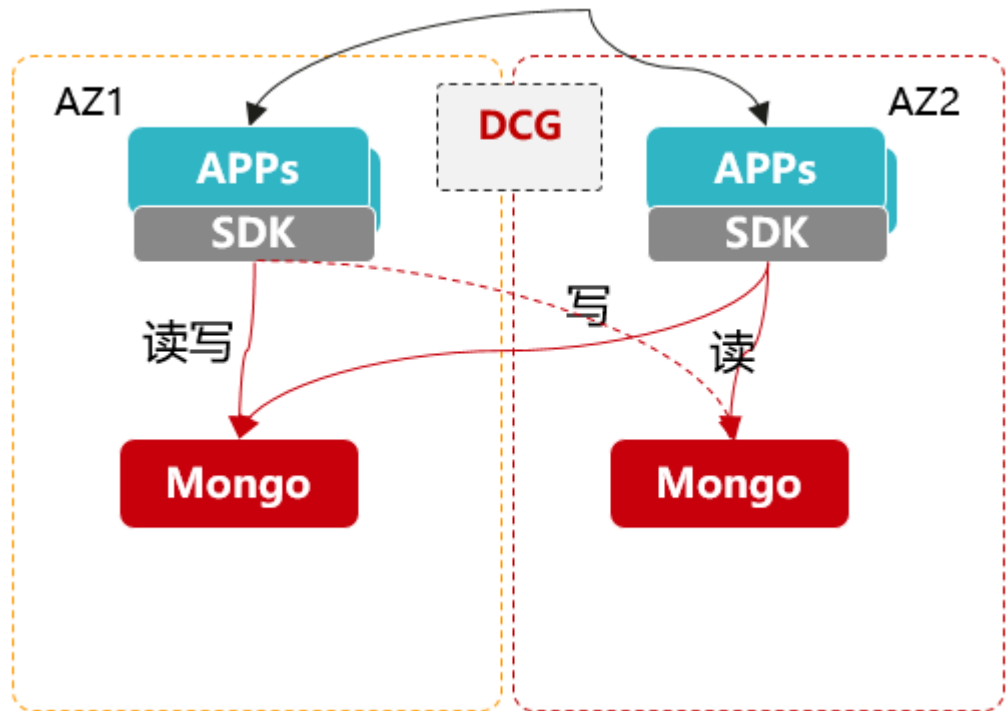
```
certificatePath: xxx #当启用https时证书路径, 实例未开启双向认证或者不启用https场景, 此项可不填
# mongo数据源配置 - 必选
sources:
  dc1:
    username: dc1user #mongo用户名
    password: dc2pwd #mongo密码
    url: mongodb://127.0.0.1:29017/mongo1 #mongo连接串(注意不能带用户名密码)
  dc2:
    username: dc2user #mongo用户名
    password: dc2pwd #mongo密码
    url: mongodb://127.0.0.1:29018/mongo1 #mongo连接串(注意不能带用户名密码)
active: dc1
routeStrategy: single-read-write #路由策略
```

4.4.2 本地读单边写

本地读单边写模式应在配置文件中配置routeStrategy为"local-read-single-write"。在这种路由模式下，读操作在本地进行，不受MAS指示的活跃节点影响。根据配置文件中的AZ信息判断本地属于哪个AZ，读操作时从本地AZ进行读取，写操作仍然根据MAS指示的活跃节点进行操作。（本地指的是当Mongo数据源和props中所配置的AZs为同一AZ时，此Mongo数据源被视为本地）

注意

- 1. 本地读单边写模式，配置文件中props.azs信息和sources.dcx.azs信息必须填写。
- 2. 如果对接MAS平台，Mongo数据源所属AZ信息，以MAS平台上的AZs信息为准，MAS平台此项选填。
- 3. 如果两个Mongo数据源的AZ信息与props.azs均相同，则会随机选取其中一个作为本地数据源进行读操作，所以需要尽量保证两个Mongo数据源的AZ信息，一个与props.azs相同，一个不同。



配置示例：

```
devspore:
dds:
# 基础信息 - 可选, 当配置etcd后必选
props:
version: v1 #项目版本号, 自定义
app-id: xxxx #应用id, 从MAS服务实例页面查询获取
monitor-id: xxxx #监控器id, 从MAS服务实例页面查询获取MAS监控id
databaseName: xxxx #MAS监控的Mongo数据库
azs: az1 #项目所属AZ, 本地读单边写模式下, 此项必填
# etcd配置, 对接MAS服务关键配置, 如不对接MAS则无需配置
etcd:
address: 127.0.0.1:2379 #etcd地址, 从MAS服务实例页面查询获取
api-version: v3 #etcd版本, v3
username: etcduser #etcd用户名, 从MAS服务实例页面查询获取
password: etcdpwd #etcd密码, 从MAS服务实例页面查询获取
https-enable: true #是否启用https, 默认为false
certificatePath: xxx #当启用https时证书路径, 实例未开启双向认证或者不启用https场景, 此项可不填
# mongo数据源配置 - 必选
sources:
dc1:
username: dc1user #mongo用户名
password: dc2pwd #mongo密码
url: mongodb://127.0.0.1:29017/mongo1 #mongo连接串(注意不能带用户名密码)
azs: az1 #此mongo数据库所属AZ
dc2:
username: dc2user #mongo用户名
password: dc2pwd #mongo密码
url: mongodb://127.0.0.1:29018/mongo1 #mongo连接串(注意不能带用户名密码)
azs: az2 #此mongo数据库所属AZ
active: dc1
routeStrategy: local-read-single-write #路由策略
```

4.4.3 强制路由

除了根据路由算法配置路由, SDK还支持通过注解`@DynamicReadRoute`指定读操作的路由。注解`@DynamicReadRoute`指定路由的参数: `source`, 用于选择数据源, 可以选择对应的枚举值: `MongoSource.ACTIVE`, `MongoSource.LOCAL`。

注解`@DynamicReadRoute`支持在方法、类上添加。

使用示例:

在方法上添加注解:

```
@DynamicRoute(source = MongoSource.ACTIVE)
public Document queryFromActive() {
    Document queryCommand =
        new Document("find", collectionName).append("filter", new Document("_id",
testExecuteCommandId));
    log.info("query command : {}", queryCommand.toJson());
    return mongoTemplate.executeCommand(queryCommand);
}
```

在类上添加注解, 此时该类的读方法都会路由到local对应的数据源上执行。

```
@DynamicRoute(source = MongoSource.LOCAL)
@Service
public class MongoService {
    @Autowired
    private MongoTemplate mongoTemplate;

    public List<RoleModel> findAll(Class<RoleModel> entityClass) {
        return mongoTemplate.findAll(entityClass);
    }

    public RoleModel insert() {
        return mongoTemplate.insert(new RoleModel("1001", "Role1001", 1001, "China1001"));
    }
}
```

```
public DeleteResult remove() {  
    return mongoTemplate.remove(new RoleModel("1011", "Role1011", 1011, "China1011"));  
}
```

4.4.4 读命令列表

MAS-Mongo-SDK将以下mongo命令视为读命令，上述的本地读单边写和注解强制路由均根据此表来操作。

表 4-1 读命令列表

序号	命令	描述
1.	aggregate、count、distinct、mapReduce	Aggregation Commands
2.	geoSearch	Geospatial Commands
3.	find	Query commands
4.	listCollections、listDatabases、listIndexes	Administration Commands

4.5 配置参数说明

表 4-2 配置参数详解

参数名称	是否必选	参数类型	取值范围	描述
props	否	PropertiesConfiguration object	请参考 表2 PropertiesConfiguration数据结构说明	MAS监控配置，配合etcd使用。
etcd	否	EtcdConfiguration object	请参考 表3 EtcdConfiguration数据结构说明	etcd配置，如配置，则会从远端拉取MongoServer配置对本地配置进行覆盖。
sources	是	ClientConfiguration object	请参考 表4 ClientConfiguration数据结构说明	MongoClient配置。
active	是	String	只能是“dc1”或“dc2”	当前使用的Mongo Server节点。
mappingConverterClassName	否	String	自定义的MappingConverter全限定类名	如com.huawei.example.MyMappingConverter。

参数名称	是否必选	参数类型	取值范围	描述
routeStrategy	否	String	single-read-write 或local-read-single-write	如不设置此配置项，默认为single-read-write。

表 4-3 PropertiesConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
version	是	String	-	项目版本号。
appld	是	String	-	MAS项目组名称。
monitorId	是	String	-	MAS监控组名称。
databaseName	是	String	-	MAS监控的Mongo数据库。
cloud	是	String	-	项目部署云组。
region	是	String	-	项目部署region。
azs	是	String	-	项目部署AZ。
decipherClassName	否	String	-	用户用于自定义加解密etcd密码的全限定类名。

表 4-4 EtcdConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
address	是	String	-	Etcd地址。
apiVersion	是	String	v3	Etcd版本，固定为v3版本。
username	是	String	-	Etcd用户名。
password	是	String	-	Etcd密码。
httpsEnable	是	Boolean	true/false	是否启用https。
certificatePath	否	String	-	证书存放目录，此目录下应包含ca.crt、client.crt、client.key.pem这三个文件。该路径支持classpath:xxx.xxx和绝对路径两种格式。

⚠ 注意

使用证书方式连接etcd时，httpsEnable必须设为true，certificatePath不可为空，且保证该路径下有相应的证书文件。

表 4-5 ClientConfiguration 数据结构说明

参数名称	是否必选	参数类型	描述
username	是	String	连接MongoDB的用户名。
password	是	String	连接MongoDB的密码。
url	是	String	MongoDB连接串(去除用户名和密码) 举例： mongodb://ip+port/database。
azs	否	String	此Mongo数据源所属AZ。当routeStrategy设置为 local-read-single-write时，此项必填。
minPoolSize	否	Integer	指定在任何时刻必须存在于单个连接池中的最小 连接数。
maxPoolSize	否	Integer	指定连接池在给定时间可以拥有的最大连接数。
waitQueueTimeoutMS	否	Integer	指定线程可以等待连接变为可用的最长时间（以 毫秒为单位）。
serverSelectionTimeoutMS	否	Integer	服务器选择超时（以毫秒为单位）。
localThresholdMS	否	Integer	当与副本集中的多个MongoDB实例进行通信时， 驱动程序只会将请求发送到响应时间小于或等于 响应时间最快的服务器加上本地阈值的服务器， 以毫秒为单位。
heartbeatFrequencyMS	否	Integer	指定驱动程序在尝试确定集群中每个服务器的当 前状态之间等待的频率（以毫秒为单位）。
replicaSet	否	String	指定提供的连接字符串包括多个主机。指定后， 驱动程序会尝试查找该集合的所有成员。
ssl	否	Boolean	指定与 MongoDB 实例的所有通信都应使用 TLS/ SSL。
tls	否	Boolean	指定与 MongoDB 实例的所有通信都应使用 TLS。
tlsInsecure	否	Boolean	指定驱动程序应允许 TLS 连接使用无效主机名。
tlsAllowInvalidHostnames	否	Boolean	指定驱动程序应允许证书中的无效主机名用于 TLS 连接。

参数名称	是否必选	参数类型	描述
connectTimeoutMS	否	Integer	指定 Java 驱动程序在超时前等待连接打开的最长时间（以毫秒为单位）。
socketTimeoutMS	否	Integer	指定 Java 驱动程序在超时之前等待发送或接收请求的最长时间（以毫秒为单位）。
maxIdleTimeMS	否	Integer	指定最长时间（以毫秒为单位），Java 驱动程序将允许池连接在关闭连接之前处于空闲状态。
maxLifeTimeMS	否	Integer	指定 Java 驱动程序在关闭连接之前将继续使用池连接的最长时间（以毫秒为单位）。
journal	否	Boolean	指定驱动程序必须等待连接的 MongoDB 实例对所有写入的磁盘上的日志文件进行分组提交。
w	否	String	指定写关注。
wtimeoutMS	否	Integer	指定写入问题的时间限制（以毫秒为单位）。
readPreference	否	String	指定读取首选。
readPreferenceTags	否	String	指定读取首选项标签。
maxStalenessSeconds	否	Integer	以秒为单位指定在驱动程序停止与辅助节点通信之前辅助节点的陈旧程度。
authMechanism	否	String	指定在提供凭据时驱动程序应使用的身份验证机制。
authSource	否	String	指定应针对所提供的凭据进行验证的数据库。
authMechanismProperties	否	String	将指定身份验证机制的身份验证属性指定为以冒号分隔的属性和值的列表。
appName	否	String	指定在连接握手期间提供给 MongoDB 实例的应用程序的名称。可用于服务器日志和分析。
compressors	否	String	指定驱动程序将尝试使用的一种或多种压缩算法来压缩发送到连接的 MongoDB 实例的请求
zlibCompressionLevel	否	String	指定 Zlib 应采用的压缩程度。
retryWrite	否	Boolean	指定如果支持的写操作由于网络错误而失败，驱动程序必须重试。默认为真。
retryRead	否	Boolean	指定如果支持的读取操作由于网络错误而失败，驱动程序必须重试。默认为真。
uuidRepresentation	否	String	指定用于读取和写入操作的 UUID 表示。

参数名称	是否必选	参数类型	描述
directConnection	否	String	指定驱动程序必须直接连接到主机。

⚠ 注意

ClientConfiguration中的url这一配置项，请勿配置MongoDB的用户名和密码。

4.6 常见问题

4.6.1 如何选择组件版本

- MAS-Mongo-SDK 1.3.1-RELEASE
springboot适用版本：2.7.0 - 2.7.2

表 4-6 核心依赖组件版本

GroupId	ArtifactId	Version	备注
org.springframework	spring-aspects	5.3.9	-
org.springframework.data	spring-data-mongo	3.4.1	-
org.mongodb	mongodb-driver-sync	4.6.1	-

- MAS-Mongo-SDK 2.1-1.3.0-RELEASE
springboot适用版本：2.1.0.RELEASE - 2.1.18.RELEASE

表 4-7 核心依赖组件版本

GroupId	ArtifactId	Version	备注
org.springframework	spring-aspects	5.3.9	-
org.springframework.data	spring-data-mongo	2.1.8.RELEASE	-

注意

org.springframework.data:spring-data-mongo 2.1.8.RELEASE, 此依赖包存在以下漏洞请参见[National Vulnerability Database](#)。

需要注意使用spring-data-mongodb的@Query和@Aggregate注解, 以防产生spel注入漏洞。

4.6.2 使用限制

- session使用

仅支持通过以下接口获取session:

```
com.huawei.devspore.mas.mongo.core.client.Cluster.startSession()  
com.huawei.devspore.mas.mongo.core.client.Cluster.startSession(ClientSessionOptions  
clientSessionOptions)  
mongoTemplate.getMongoDbFactory().getSession(ClientSessionOptions.builder().build())
```

此session对应的Mongo数据源为当前active数据源。所以在使用session相关的操作时, 请务必保证所有操作都能路由到active数据源。

在本地读单边写模式下, 如下使用方式会抛出异常:

java.lang.IllegalStateException: state should be: ClientSession from same
MongoClient

```
SessionScoped sessionScoped = mongoTemplate.withSession(ClientSessionOptions.builder().build());  
sessionScoped.execute(action -> {  
    action.insert(xxx); // 写操作, 路由到active数据源  
    action.count(xxx); // 读操作, 路由到本地数据源  
});
```

- 2. 有限制的命令

表 4-8 MAS-Mongo-SDK 有限制的命令

命令名称	使用限制	原因
MongoTemplate.findAll AndRemove()	本地读单边写模式下, 此命令不支持。	原生mongo没有此命令, 是spring-data-mongo对find和remove 操作的封装。

5 MAS-GO-SDK 使用手册

[5.1 MySQL](#)

[5.2 Redis](#)

[5.3 故障注入](#)

5.1 MySQL

5.1.1 概述

5.1.1.1 开发简介

本文主要描述如何使用MAS-GO-SDK在多活容灾场景下对涉及MySQL的服务进行开发，结合样例讲解在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Go语言，并有Go程序开发经验。
- 熟悉module开发模式。
- 熟悉MySQL的常用操作。

MAS-GO-SDK-MySQL

MAS-GO-SDK-MySQL是基于go-sql-driver/mysql v1.6.0开发的实现支持多活容灾服务的MySQL驱动源，其宗旨是促进开发者对于异地多活的MySQL关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

5.1.1.2 开发流程

开发的流程如下所示：

1. 版本获取及引入依赖。
通过go.mod引入需要的依赖，是使用MAS-GO-SDK的基础。
2. 配置创建数据源。

通过配置数据源驱动信息，引入MAS-GO-SDK-MySQL创建数据源。
支持以代码方式添加配置信息或者从YAML文件读取配置信息。

3. 按需引入数据源。

在需要使用数据源的地方引入并执行MySQL操作。

5.1.2 环境准备

在进行应用开发时，准备环境包括[开发环境准备](#)所示任务。

表 5-1 开发环境准备

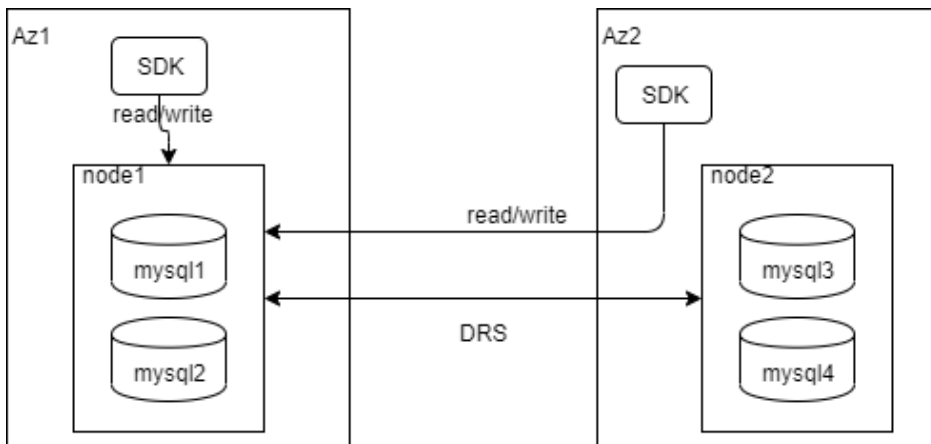
准备项	说明
准备操作系统	Windows系统，推荐使用Windows 7及以上版本。
安装Go	开发环境的基本配置。Go版本要求1.14.6及以上。
安装配置Goland	开发程序的工具。Goland使用11.0及以上版本。

5.1.3 使用场景

本模块数据源创建支持single-read-write（单读写），local-read-single-write（本地读单写）2种模式，同时SDK内置实现了读写分离，可通过配置RANDOM（随机）或ROUND_ROBIN（轮询）负载均衡算法，搭配MAS可实现多活容灾；同时内置故障注入功能可创建带有注入故障的实例，进行相关业务场景的模拟，配置修改请参考[故障注入MySQL配置示例](#)。

该模块具有如下特性：

- 多活容灾能力。
多活容灾能力是指在同城场景下实现多活故障自动切换，由SDK和MAS服务配合完成。
- 读写分离。
读写分离由SDK实现，支持随机、轮询的负载均衡算法。
- 故障注入。
支持带注入故障的实例创建，包含延时（波动）、预设异常的故障配置。
 - single-read-write
单读写类似于主备库，MAS监控各节点健康状态，active节点故障自动切换至其他节点，保证多活容灾能力，节点之间用DRS进行数据同步，保证数据一致。

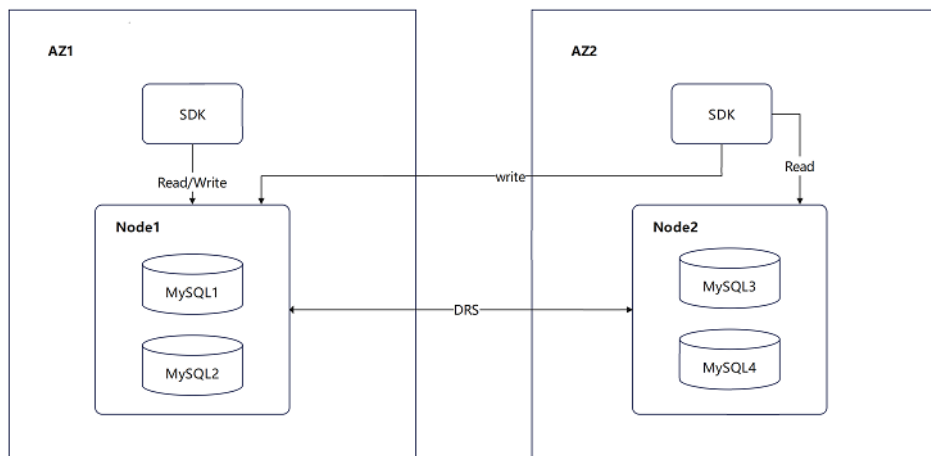


读操作：同步路由到active的mysql。

写操作：同步路由到active的mysql。

- local-read-single-write

本地读单写适用于读多写少场景，读操作会根据路由算法读取近端节点，写操作遵循写入至active节点，MAS监控各节点健康状态，active节点故障自动切换至其他节点，节点之间用DRS进行数据同步。



读操作：同步路由到近端mysql。

写操作：同步路由到active的mysql。

5.1.4 使用指南

5.1.4.1 原生 DB

创建go版本下database/sql包sql.DB数据源，进行mysql操作。

步骤1 在go.mod文件中引入依赖，即引入MAS-GO-SDK。

github.com/huaweicloud/devcloud-go

```
go.mod x
1  module demo
2
3  go 1.14
4  require (
5      github.com/huaweicloud/devcloud-go v0.1.1
6  )
```

步骤2 结合[配置项说明](#)，添加配置文件或者配置信息。

- 配置文件示例config_mysql.yaml。

```
props:
  version: v1
  appld: xxx
  monitorId: xxxx
  databaseName: xxxx
etcd:
  address: xxx.xxx.xxx.xxx:xxxx
  apiVersion: v3
  username: xxxx
  password: xxxx
  httpsEnable: false
# require
datasource:
  ds0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0
    username: xxxx
    password: xxxx
  ds0-slave0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave0
    username: xxxx
    password: xxxx
  ds0-slave1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave1
    username: xxxx
    password: xxxx
  ds1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1
    username: xxxx
    password: xxxx
  ds1-slave0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave0
    username: xxxx
    password: xxxx
  ds1-slave1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave1
    username: xxxx
    password: xxxx
# require
router:
  active: c0
  routeAlgorithm: single-read-write
  retry:
    times: 10
    delay: 50# ms
  nodes:
    c0:
      master: ds0
      loadBalance: RANDOM
      slaves:
        - ds0-slave0
        - ds0-slave1
    c1:
      master: ds1
```

```
loadBalance: ROUND_ROBIN
slaves:
- ds1-slave0
- ds1-slave1
```

- 配置信息。

```
func mysqlConfiguration() *config.ClusterConfiguration {
return &config.ClusterConfiguration{
Props: &mas.PropertiesConfiguration{
Version: "v1",
ApplID: "xxx",
MonitorID: "xxxx",
DatabaseName: "xxx",
},
EtcdConfig: &etcd.EtcdConfiguration{
Address: "xxx.xxx.xxx.xxx:xxxx",
APIVersion: "v3",
Username: "xxxx",
Password: "xxxx",
HTTPSEnable: false,
},
RouterConfig: &config.RouterConfiguration{
Nodes: map[string]*config.NodeConfiguration{
"dc0": {
Master: "ds0",
LoadBalance: "RANDOM",
Slaves: []string{"ds0-slave0", "ds0-slave1"},
},
"dc1": {
Master: "ds1",
LoadBalance: "ROUND_ROBIN",
Slaves: []string{"ds1-slave0", "ds1-slave1"},
},
},
Active: "dc0",
Retry: &config.RetryConfiguration{
Times: "10",
Delay: "50",
},
RouteAlgorithm: "single-read-write",
},
DataSource: map[string]*config.DataSourceConfiguration{
"ds0": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0",
Username: "xxxx",
Password: "xxxx",
},
"ds0-slave0": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave0",
Username: "xxxx",
Password: "xxxx",
},
"ds0-slave1": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave1",
Username: "xxxx",
Password: "xxxx",
},
"ds1": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1",
Username: "xxxx",
Password: "xxxx",
},
"ds1-slave0": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave0",
Username: "xxxx",
Password: "xxxx",
},
"ds1-slave1": {
URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave1",
Username: "xxxx",

```

```
        Password: "xxxx",
    },
},
}
```

步骤3 初始化创建sql.DB数据源DevSporeDb。

- 配置文件方式创建

```
import (
    "database/sql"
    "log"

    _ "github.com/huaweicloud/devcloud-go/sql-driver/mysql"
)
var (
    DevSporeDb *sql.DB
    err error
)
func init() {
    DevSporeDb, err = sql.Open("devspore_mysql", "./conf/resources/config_mysql.yaml")
    if err != nil {
        log.Fatalln(err)
    }
}
```

- 配置信息方式创建。

```
import (
    "database/sql"
    "log"

    devspore "github.com/huaweicloud/devcloud-go/sql-driver/mysql"
)
var (
    DevSporeDb *sql.DB
    err error
)
func init() {
    devspore.SetClusterConfiguration(mysqlConfiguration())
    DevSporeDb, err = sql.Open("devspore_mysql", "")

    if err != nil {
        log.Fatalln(err)
    }
}
```

步骤4 在需要执行MySQL操作的地方使用DevSporeDb执行MySQL操作，具体执行因业务各异，执行对应的sql.DB命令。

----结束

5.1.4.2 beego-orm

创建github.com/astaxie/beego/orm包**orm.Ormer**数据源，进行mysql操作。

依赖引入，配置文件或配置信息修改参考[原生DB](#)。

初始化创建beego-orm数据源DevSporeOrm，后续执行MySQL相关操作。

由于beego-orm需要注册使用的model，以Teacher，Student为例。

```
type Teacher struct {
    Id int
    Name string
    Age int
}
type Student struct {
    Id int
    Name string
```

```
Age int
}
```

- 配置文件方式创建。

```
import (
    "log"

    "github.com/astaxie/beego/orm"
    _ "github.com/haaweicloud/devcloud-go/sql-driver/mysql"
)
var (
    DevSporeOrm orm.Ormer
    err error
)
func init() {
    // 1 注册devspore_mysql
    err = orm.RegisterDriver("devspore_mysql", orm.DRMySQL)
    if err != nil {
        log.Fatalln(err)
    }
    // 2 注册使用model
    orm.RegisterModel(new(Teacher),new(Student))
    // 3 创建数据源
    err = orm.RegisterDataBase("default", "devspore_mysql", "./conf/resources/config_mysql.yaml")
    if err != nil {
        log.Fatalln(err)
    }
    DevSporeOrm = orm.NewOrm()
}
```

- 配置信息方式创建。

```
import (
    "log"

    "github.com/astaxie/beego/orm"
    devspore "github.com/haaweicloud/devcloud-go/sql-driver/mysql"
)
var (
    DevSporeOrm orm.Ormer
    err error
)
func init() {
    // 1 注册devspore_mysql
    err = orm.RegisterDriver("devspore_mysql", orm.DRMySQL)
    if err != nil {
        log.Fatalln(err)
    }
    // 2 注册使用model
    orm.RegisterModel(new(Teacher),new(Student))
    // 3 创建数据源
    devspore.SetClusterConfiguration(mysqlConfiguration())
    err = orm.RegisterDataBase("default", "devspore_mysql", "")
    if err != nil {
        log.Fatalln(err)
    }
    DevSporeOrm = orm.NewOrm()
}
```

5.1.4.3 gorm

创建gorm.io/gorm包gorm.DB数据源，进行mysql操作。

依赖引入，配置文件或配置信息修改参考[原生DB](#)。

初始化创建gorm数据源DevSporeGorm，后续执行MySQL相关操作。

- 配置文件方式创建。

```
import (
    "log"
```

```

_ "github.com/huaweicloud/devcloud-go/sql-driver/mysql"
"gorm.io/driver/mysql"
"gorm.io/gorm"
)
var (
    DevSporeGorm *gorm.DB
    err error
)
func init() {
    DevSporeGorm, err = gorm.Open(mysql.New(
        mysql.Config{DriverName: "devspore_mysql", DSN: "./conf/resources/config_mysql.yaml"},
    ))
    if err != nil {
        log.Fatalln(err)
    }
}
}

```

- 配置信息方式创建。

```

import (
    "log"

    devspore "github.com/huaweicloud/devcloud-go/sql-driver/mysql"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
)
var (
    DevSporeGorm *gorm.DB
    err error
)
func init() {
    devspore.SetClusterConfiguration(mysqlConfiguration())
    DevSporeGorm, err = gorm.Open(mysql.New(mysql.Config{DriverName: "devspore_mysql", DSN:
""}))
    if err != nil {
        log.Fatalln(err)
    }
}
}

```

5.1.5 配置项说明

表 5-2 配置参数详解

参数名称	是否必选	参数类型	取值范围	描述
props	否	PropertiesConfiguration	请参考 PropertiesConfiguration数据结构说明 。	MAS监控配置，配合etcd使用。
etcd	否	EtcdConfiguration	请参考 EtcdConfiguration数据结构说明 。	etcd配置，如配置，则会从远端拉取。
datasource	是	map[string]DataSourceConfiguration	key自定义，单维度请参考 表 5-3 。	数据源。

参数名称	是否必选	参数类型	取值范围	描述
router	是	RouterConfiguration	请参考 RouterConfiguration数据结构说明 。	路由相关配置。
chaos	否	InjectionProperties	请参考 InjectionProperties数据结构说明 。	故障注入相关配置。

表 5-3 DataSourceConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
url	是	string	protocol(address)/dbname?param=value	Data Source Name, 数据源连接串。
username	是	string	-	用户名。
password	是	string	-	密码。

表 5-4 RouterConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
active	是	string	nodes的key	激活节点。
routeAlgorithm	是	string	single-read-write local-read-single-write	路由算法。
retry.times	否	string	-	失败重试次数。
retry.delay	否	string	-	重试间隔, 单位毫秒。
nodes	是	map[string]NodeConfiguration	key自定义, 单维度参考 NodeConfiguration数据结构说明	节点相关配置。

表 5-5 NodeConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
master	是	string	datasource的key	主节点数据源。
loadBalance	否	string	RANDOM、ROUND_ROBIN	读写分离负载均衡算法。
slaves	否	[]string	datasource的key	从节点数据源。

表 5-6 PropertiesConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
version	是	string	-	项目版本号。
appld	是	string	-	项目组名称。
monitorId	是	string	-	监控组名称。
databaseName	是	string	-	数据库名称。

表 5-7 EtcConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
address	是	string	-	Etc地址。
apiVersion	是	string	v3	Etc版本。
username	是	string	-	Etc用户名。
password	是	string	-	Etc密码。
httpsEnable	是	bool	true/false	是否启用https。

5.2 Redis

5.2.1 概述

5.2.1.1 开发简介

本文主要描述如何使用MAS-GO-SDK在多活容灾场景下对涉及Redis的服务进行开发，结合样例讲解在开发过程中如何使用。

本文假设您已经具备如下开发能力：

- 熟悉Go语言，并有Go程序开发经验。
- 熟悉module开发模式。
- 熟悉Redis的常用操作。

MAS-GO-SDK-Redis

MAS-GO-SDK-Redis是一个在go-redis v8.11.3的基础上实现的支持多活容灾服务的Redis连接客户端，其宗旨是促进开发者对于异地多活的Redis关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

5.2.1.2 开发流程

开发的流程如下所示：

1. 版本获取及引入依赖
通过go.mod引入需要的依赖，是使用MAS-GO-SDK的基础。
2. 配置创建DevsporeClient客户端
通过配置客户端信息，引入MAS-GO-SDK-Redis创建DevsporeClient客户端。
支持以代码方式添加配置信息或者从YAML文件读取配置信息。
3. 按需引入DevsporeClient客户端
在需要使用DevsporeClient客户端的地方引入并执行Redis操作。

5.2.2 环境准备

在进行应用开发时，准备环境包括[开发环境准备](#)所示任务。

表 5-8 开发环境准备

准备项	说明
准备操作系统	Windows系统，推荐使用Windows 7及以上版本。
安装Go	开发环境的基本配置。版本要求1.14.6及以上。
安装配置Goland	Goland使用11.0及以上版本，用于开发程序的工具。

5.2.3 使用场景

本模块客户端创建支持single-read-write（单读写），local-read-single-write（本地读单写），double-write（本地读双写）3种模式，其中Redis节点可配置cluster（集

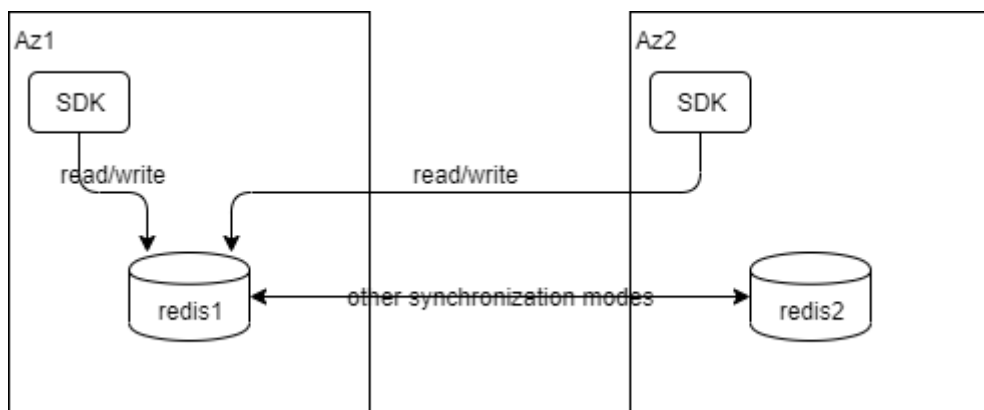
群), master-slave (主从), normal (普通) Redis服务, 搭配MAS可实现多活容灾; 同时内置故障注入功能可创建带有注入故障的实例, 进行相关业务场景的模拟, 配置修改请参考[故障注入Redis配置示例](#)。

该模块具有如下特性:

- 多活容灾能力
多活容灾能力是指在同城场景下实现多活故障自动切换, 由SDK和MAS服务配合完成。
- 数据双写
读写分离由SDK实现, 支持内存双写、文件双写。
- 故障注入
支持带注入故障的实例创建, 包含延时(波动)、预设异常的故障配置。

1. single-read-write

单读写类似于主备库, MAS监控各节点健康状态, active节点故障自动切换至其他节点, 保证多活容灾能力, 节点之间进行数据同步, 保证数据一致。

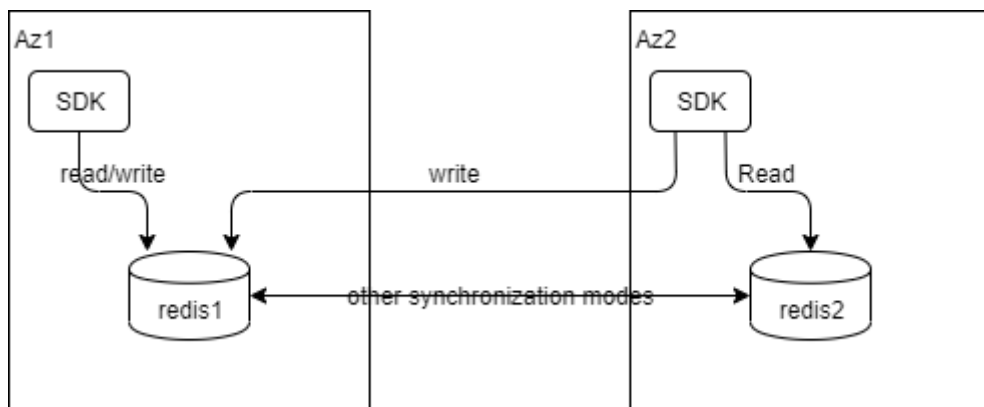


读操作: 同步路由到active的Redis。

写操作: 同步路由到active的Redis。

2. local-read-single-write

本地读单写适用于读多写少场景, 读操作会根据路由算法读取近端节点, 写操作遵循写入至active节点, MAS监控各节点健康状态, active节点故障自动切换至其他节点, 节点之间进行数据同步。

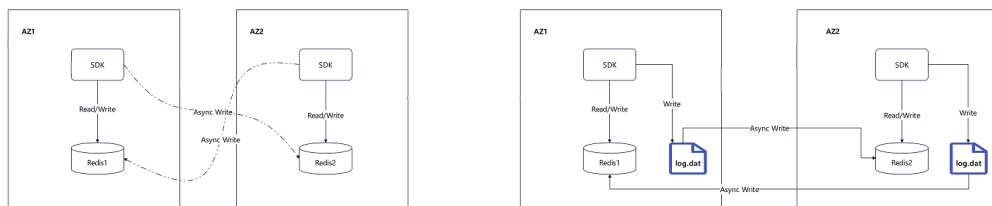


读操作: 同步路由到近端Redis。

写操作: 同步路由到active的Redis。

3. double-write

本地读双写同样适用于读多写少场景，读操作直接读取近端节点，写操作在近端执行完会异步写到远端节点，不需要外部进行数据同步。



读操作：同步路由到近端Redis。

写操作：

- 内存双写：同步路由到近端Redis，同时异步发送到远端Redis。
- 文件双写：同步路由到近端Redis，同时异步写文件，定时扫描发送至远端Redis。

5.2.4 使用指南

步骤1 在go.mod文件中引入依赖，即引入MAS-GO-SDK。

github.com/huaweicloud/devcloud-go

```

1  module demo
2
3  go 1.14
4  require (
5      github.com/huaweicloud/devcloud-go v0.1.1
6  )

```

步骤2 结合[配置项说明](#)，添加配置文件或者配置信息。

- 配置文件示例config_Redis.yaml

```

props:
  version: v1
  appld: xxx
  monitorId: xxxx
  cloud: xxxx
  region: xxxxx
  azs: az1
etcd:
  address: xxx.xxx.xxx.xxx:xxxx
  apiVersion: v3
  username: xxx
  password: xxx
  httpsEnable: false
Redis:
  nearest: dc1
  asyncRemoteWrite:
    retryTimes: 4
  connectionPool:
    enable: true
  asyncRemotePool:
    persist: true
  threadCoreSize: 10
  taskQueueSize: 5
  persistDir: dataDir/

```

```
servers:
  dc1:
    hosts: xxx.xxx.xxx.xxx:xxxx
    password: xxxxxx
    type: normal# cluster, master-slave, normal
    cloud: xxxx
    region: xxxxx
    azs: az1
    pool:
      maxTotal: 100
      maxIdle: 8
      minIdle: 0
      maxWaitMillis: 10000
      timeBetweenEvictionRunsMillis: 1000
  dc2:
    hosts: xxx.xxx.xxx.xxx:xxxx, xxx.xxx.xxx.xxx:xxxx
    password: xxxxxx
    type: cluster# cluster, master-slave, normal
    cloud: xxxx
    region: xxxxx
    azs: az1
    pool:
      maxTotal: 100
      maxIdle: 8
      minIdle: 0
      maxWaitMillis: 10000
      timeBetweenEvictionRunsMillis: 1000
routeAlgorithm: single-read-write# local-read-single-write, single-read-write, double-write
active: dc1
```

- 配置信息

```
func RedisConfiguration() *config.Configuration {
  return &config.Configuration{
    Props: &mas.PropertiesConfiguration{
      Version: "v1",
      ApplID: "xxx",
      MonitorID: "xxxx",
      Cloud: "xxx",
      Region: "xxxxx",
      Azs: "az1",
    },
    EtcdConfig: &etcd.EtcdConfiguration{
      Address: "xxx.xxx.xxx.xxx:xxxx",
      APIVersion: "v3",
      Username: "xxxx",
      Password: "xxxx",
      HTTPSEnable: false,
    },
    RedisConfig: &config.RedisConfiguration{
      Nearest: "dc1",
      AsyncRemoteWrite: &config.AsyncRemoteWrite{
        RetryTimes: 4,
      },
      ConnectionPoolConfig: &config.RedisConnectionPoolConfiguration{
        Enable: true,
      },
      AsyncRemotePoolConfiguration: &config.AsyncRemotePoolConfiguration{
        Persist: true,
        ThreadCoreSize: 10,
        TaskQueueSize: 5,
        PersistDir: "dataDir/",
      },
    },
    Servers: map[string]*config.ServerConfiguration{
      "dc1": {
        Hosts: "xxx.xxx.xxx.xxx:xxxx",
        Password: "xxxxxx",
        Type: "normal",
        Cloud: "xxxx",
        Region: "xxxx",
        Azs: "az1",
      },
    },
  }
}
```

```
    ConnectionPool: &config.ServerConnectionPoolConfiguration{
      MaxTotal: 100,
      MaxIdle: 8,
      MinIdle: 0,
      MaxWaitMillis: 10000,
      TimeBetweenEvictionRunsMillis: 1000,
    },
  },
  "dc2": {
    Hosts: "xxx.xxx.xxx.xxx:xxxx,xxx.xxx.xxx.xxx:xxxx",
    Password: "xxxxxx",
    Type: "cluster",
    Cloud: "xxx",
    Region: "xxx",
    Azs: "az1",
    ConnectionPool: &config.ServerConnectionPoolConfiguration{
      MaxTotal: 100,
      MaxIdle: 8,
      MinIdle: 0,
      MaxWaitMillis: 10000,
      TimeBetweenEvictionRunsMillis: 1000,
    },
  },
},
RouteAlgorithm: "single-read-write",
Active: "dc1",
}
}
```

步骤3 初始化创建DevsporeClient客户端。

- 配置文件方式创建

```
import (
    "context"
    "log"

    "github.com/huaweicloud/devcloud-go/redis"
)
var (
    DevSporeClient *redis.DevsporeClient // Redis 客户端
)
func init() {
    DevSporeClient = redis.NewDevsporeClientWithYaml("./resources/config_Redis.yaml")
    if_, err := DevSporeClient.Ping(context.Background()).Result(); err != nil {
        log.Fatal(err)
    }
    log.Println("INFO: init redis succeeded.")
}
```

- 配置信息方式创建

```
import (
    "context"
    "log"

    "github.com/huaweicloud/devcloud-go/redis"
)
var (
    DevSporeClient *redis.DevsporeClient // redis 客户端
)
func init() {
    DevSporeClient = redis.NewDevsporeClient(RedisConfiguration())
    if_, err := DevSporeClient.Ping(context.Background()).Result(); err != nil {
        log.Fatal(err)
    }
    log.Println("INFO: init redis succeeded.")
}
```

步骤4 在需要执行Redis操作的地方使用DevsporeClient执行Redis操作。

```

import (
    "context"
    "demo/conf"
    "time"

    "github.com/go-redis/redis/v8"
)
func set(ctx context.Context, key, val string, time time.Duration) *redis.StatusCmd {
    return conf.DevSporeClient.Set(ctx, key, val, time)
}
func get(ctx context.Context, key string) *redis.StringCmd {
    return conf.DevSporeClient.Get(ctx, key)
}

```

----结束

5.2.5 配置项说明

表 5-9 配置参数详解

参数名称	是否必选	参数类型	取值范围	描述
props	否	PropertiesConfiguration	请参考 PropertiesConfiguration数据结构说明 。	MAS监控配置，配合etcd使用。
etcd	否	EtcdConfiguration	请参考 EtcdConfiguration数据结构说明 。	etcd配置，如配置，则会从远端拉取。
redis	是	RedisConfiguration	请参考 RedisConfiguration数据结构说明 。	RedisServer配置。
routeAlgorithm	是	string	<ul style="list-style-type: none"> • single-read-write • local-read-single-write • double-write 	路由算法。
active	是	string	只能是“dc1”或“dc2”。	激活的Redis。
chaos	否	InjectionProperties	请参考 InjectionProperties数据结构说明 。	故障注入相关配置。

表 5-10 RedisConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
nearest	否	string	只能是“dc1”或“dc2”。	指明哪个是近端Redis。
asyncRemoteWrite.retryTimes	否	int	默认为3。	异步写远端操作重试次数。
connectionPool.enable	否	bool	true/false默认true。	是否启用连接池。
asyncRemotePool	否	AsyncRemotePoolConfiguration	请参考 AsyncRemotePoolConfiguration数据结构说明 。	异步写线程池配置。
servers	是	map[string]ServerConfiguration	key为dc1/dc2 单个维度请参考 ServerConfiguration数据结构说明 。	dc1, dc2的RedisServer连接配置。

表 5-11 AsyncRemotePoolConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
threadCoreSize	否	int	-	线程池的基本大小。
persist	否	bool	true/false默认false。	命令是否持久化，否：速度快；是：速度比非持久化低。
taskQueueSize	否	int	默认5。	缓冲队列数。
persistDir	否	string	默认根目录"/"。	Redis日志文件目录。

表 5-12 ServerConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
hosts	是	string	-	RedisServer地址。

参数名称	是否必选	参数类型	取值范围	描述
password	是	string	-	RedisServer密码。
type	是	string	cluster, master-slave, normal。	RedisServer类型。
cloud	是	string	-	RedisServer所属云。
region	是	string	-	RedisServer所属Region。
azs	是	string	-	RedisServer所属AZ。
pool	否	ServerConnection PoolConfiguration	请参考 ServcerConnectionPoolConfiguration 数据结构说明。	连接池配置。

表 5-13 ServcerConnectionPoolConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
maxTotal	否	int	-	最大活动对象数。
maxIdle	否	int	-	最大能够保持idle状态的对象数。
minIdle	否	int	-	最小能够保持idle状态的对象数。
maxWaitMillis	否	int	-	当池内没有返回对象时，最大等待时间。
timeBetweenEvictionRunsMillis	否	int	-	空闲连接检测线程，检测的周期，毫秒数。如果为负值，表示不运行检测线程。默认为-1。

表 5-14 PropertiesConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
version	是	string	-	项目版本号。

参数名称	是否必选	参数类型	取值范围	描述
appld	是	string	-	项目组名称。
monitorId	是	string	-	监控组名称。
cloud	否	string	-	项目部署云组。
region	否	string	-	项目部署region。
azs	否	string	-	项目部署AZ。

表 5-15 EtcdConfiguration 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
address	是	string	-	Etcd地址。
apiVersion	是	string	v3	Etcd版本。
username	是	string	-	Etcd用户名。
password	是	string	-	Etcd密码。
httpsEnable	是	bool	true/false	是否启用https。

5.3 故障注入

5.3.1 概述

故障注入功能是在上述模块的补充功能，可在对应服务添加故障注入配置创建带有注入故障的实例，可注入带波动的延时故障和异常故障，进行相关业务场景的模拟。

具体操作请参考[MySQL配置示例](#)和[Redis配置示例](#)。

相关配置详见[配置项说明](#)和[内置注入故障](#)。

5.3.2 使用指南

5.3.2.1 MySQL 配置示例

结合[配置项说明](#)，在配置文件或配置信息添加故障注入相关配置。

- 配置文件示例config_mysql_chaos.yamll

```
props:
  version: v1
  appld: xxx
  monitorId: xxxx
```

```
databaseName: xxxx
etcd:
  address: xxx.xxx.xxx.xxx:xxxx
  apiVersion: v3
  username: xxxx
  password: xxxx
  httpsEnable: false
# require
datasource:
  ds0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0
    username: xxxx
    password: xxxx
  ds0-slave0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave0
    username: xxxx
    password: xxxx
  ds0-slave1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave1
    username: xxxx
    password: xxxx
  ds1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1
    username: xxxx
    password: xxxx
  ds1-slave0:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave0
    username: xxxx
    password: xxxx
  ds1-slave1:
    url: tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave1
    username: xxxx
    password: xxxx
# require
router:
  active: c0
  routeAlgorithm: single-read-write
  retry:
    times: 10
    delay: 50# ms
  nodes:
    c0:
      master: ds0
      loadBalance: RANDOM
      slaves:
        - ds0-slave0
        - ds0-slave1
    c1:
      master: ds1
      loadBalance: ROUND_ROBIN
      slaves:
        - ds1-slave0
        - ds1-slave1
chaos:
  active: true# 全局开关 默认false
  duration: 20
  interval: 100
  percentage: 100
  delayInjection:
    active: true
    percentage: 75
    timeMs: 1000
    jitterMs: 500
  errorInjection:
    active: true
    percentage: 20
```

- 配置信息

```
func mysqlConfiguration() *config.ClusterConfiguration {
    return &config.ClusterConfiguration{
        Props: &mas.PropertiesConfiguration{
            Version: "v1",
            AppID: "xxx",
            MonitorID: "xxxx",
            DatabaseName: "xxx",
        },
        EtcdConfig: &etcd.EtcdConfiguration{
            Address: "xxx.xxx.xxx.xxx:xxxx",
            APIVersion: "v3",
            Username: "xxxx",
            Password: "xxxx",
            HTTPSEnable: false,
        },
        RouterConfig: &config.RouterConfiguration{
            Nodes: map[string]*config.NodeConfiguration{
                "dc0": {
                    Master: "ds0",
                    LoadBalance: "RANDOM",
                    Slaves: []string{"ds0-slave0", "ds0-slave1"},
                },
                "dc1": {
                    Master: "ds1",
                    LoadBalance: "ROUND_ROBIN",
                    Slaves: []string{"ds1-slave0", "ds1-slave1"},
                },
            },
            Active: "dc0",
            Retry: &config.RetryConfiguration{
                Times: "10",
                Delay: "50",
            },
            RouteAlgorithm: "single-read-write",
        },
        DataSource: map[string]*config.DataSourceConfiguration{
            "ds0": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0",
                Username: "xxxx",
                Password: "xxxx",
            },
            "ds0-slave0": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave0",
                Username: "xxxx",
                Password: "xxxx",
            },
            "ds0-slave1": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds0-slave1",
                Username: "xxxx",
                Password: "xxxx",
            },
            "ds1": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1",
                Username: "xxxx",
                Password: "xxxx",
            },
            "ds1-slave0": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave0",
                Username: "xxxx",
                Password: "xxxx",
            },
            "ds1-slave1": {
                URL: "tcp(xxx.xxx.xxx.xxx:xxxx)/ds1-slave1",
                Username: "xxxx",
                Password: "xxxx",
            },
        },
        Chaos: &mas.InjectionProperties{
            Active: true,
        },
    }
}
```

```
    Duration: 20,
    Interval: 100,
    Percentage: 100,
    DelayInjection: &mas.DelayInjection{
      Active: true,
      Percentage: 75,
      TimeMs: 1000,
      JitterMs: 500,
    },
    ErrorInjection: &mas.ErrorInjection{
      Active: true,
      Percentage: 20,
    },
  },
}
```

5.3.2.2 Redis 配置示例

结合[配置项说明](#)，在配置文件或配置信息添加故障注入相关配置。

- 配置文件示例config_Redis_chaos.yaml

```
props:
  version: v1
  appld: xxx
  monitorId: xxxx
  cloud: xxxx
  region: xxxxx
  azs: az1
etcd:
  address: xxx.xxx.xxx.xxx:xxxx
  apiVersion: v3
  username: xxxx
  password: xxxx
  httpsEnable: false
Redis:
  nearest: dc1
  asyncRemoteWrite:
    retryTimes: 4
  connectionPool:
    enable: true
  asyncRemotePool:
    persist: true
    threadCoreSize: 10
    taskQueueSize: 5
    persistDir: dataDir/
servers:
  dc1:
    hosts: xxx.xxx.xxx.xxx:xxxx
    password: xxxxxx
    type: normal# cluster, master-slave, normal
    cloud: xxxx
    region: xxxxx
    azs: az1
    pool:
      maxTotal: 100
      maxIdle: 8
      minIdle: 0
      maxWaitMillis: 10000
      timeBetweenEvictionRunsMillis: 1000
  dc2:
    hosts: xxx.xxx.xxx.xxx:xxxx, xxx.xxx.xxx.xxx:xxxx
    password: xxxxxx
    type: cluster# cluster, master-slave, normal
    cloud: xxxx
    region: xxxxx
    azs: az1
    pool:
```

```
maxTotal: 100
maxIdle: 8
minIdle: 0
maxWaitMillis: 10000
timeBetweenEvictionRunsMillis: 1000
routeAlgorithm: single-read-write# local-read-single-write, single-read-write, double-write
active: dc1
chaos:
  active: true# 全局开关 默认false
  duration: 20
  interval: 100
  percentage: 100
  delayInjection:
    active: true
    percentage: 75
    timeMs: 1000
    jitterMs: 500
  errorInjection:
    active: true
    percentage: 20
```

- 配置信息

```
func RedisConfiguration() *config.Configuration {
  return &config.Configuration{
    Props: &mas.PropertiesConfiguration{
      Version: "v1",
      AppID: "xxx",
      MonitorID: "xxxx",
      Cloud: "xxx",
      Region: "xxxxx",
      Azs: "az1",
    },
    EtcConfig: &etcd.EtcdConfiguration{
      Address: "xxx.xxx.xxx.xxx:xxxx",
      APIVersion: "v3",
      Username: "xxxx",
      Password: "xxxx",
      HTTPSEnable: false,
    },
    RedisConfig: &config.RedisConfiguration{
      Nearest: "dc1",
      AsyncRemoteWrite: &config.AsyncRemoteWrite{
        RetryTimes: 4,
      },
      ConnectionPoolConfig: &config.RedisConnectionPoolConfiguration{
        Enable: true,
      },
      AsyncRemotePoolConfiguration: &config.AsyncRemotePoolConfiguration{
        Persist: true,
        ThreadCoreSize: 10,
        TaskQueueSize: 5,
        PersistDir: "dataDir/",
      },
    },
    Servers: map[string]*config.ServerConfiguration{
      "dc1": {
        Hosts: "xxx.xxx.xxx.xxx:xxxx",
        Password: "xxxxxx",
        Type: "normal",
        Cloud: "xxxx",
        Region: "xxxx",
        Azs: "az1",
        ConnectionPool: &config.ServerConnectionPoolConfiguration{
          MaxTotal: 100,
          MaxIdle: 8,
          MinIdle: 0,
          MaxWaitMillis: 10000,
          TimeBetweenEvictionRunsMillis: 1000,
        },
      },
    },
  },
}
```

```
"dc2": {
  Hosts: "xxx.xxx.xxx.xxx:xxxx,xxx.xxx.xxx.xxx:xxxx",
  Password: "xxxxxx",
  Type: "cluster",
  Cloud: "xxx",
  Region: "xxx",
  Azs: "az1",
  ConnectionPool: &config.ServerConnectionPoolConfiguration{
    MaxTotal: 100,
    MaxIdle: 8,
    MinIdle: 0,
    MaxWaitMillis: 10000,
    TimeBetweenEvictionRunsMillis: 1000,
  },
},
},
},
RouteAlgorithm: "single-read-write",
Active: "dc1",
Chaos: &mas.InjectionProperties{
  Active: true,
  Duration: 20,
  Interval: 100,
  Percentage: 100,
  DelayInjection: &mas.DelayInjection{
    Active: true,
    Percentage: 75,
    TimeMs: 1000,
    JitterMs: 500,
  },
  ErrorInjection: &mas.ErrorInjection{
    Active: true,
    Percentage: 20,
  },
},
}
```

5.3.3 配置项说明

表 5-16 InjectionProperties 数据结构说明

参数名称	是否必选	参数类型	取值范围	描述
active	是	bool	true/false 默认false	故障注入功能是否开启。
duration	是	int	-	故障注入持续时间，单位：秒。
interval	是	int	-	故障注入间隔时间，单位：秒。
percentage	是	int	0-100	注入故障概率。
delayInjection.active	是	bool	true/false	延时注入开关。
delayInjection.percentage	是	int	0-100	延时故障生效概率。

参数名称	是否必选	参数类型	取值范围	描述
delayInjection.timeMs	是	int	-	延时基数，单位：毫秒。
delayInjection.jitterMs	是	int	-	延时抖动幅度，单位：毫秒。
errorInjection.active	是	bool	true/false	异常注入开关。
errorInjection.percentage	是	int	0-100	异常故障生效概率。

5.3.4 内置注入故障

- 带波动的延时故障
基于延时基数和抖动幅度创建带有波动的延时故障。具体触发要结合故障注入功能是否开启、注入故障概率、延时注入开关、延时故障生效概率决定。
- 异常故障
内置如下异常故障，结合故障注入功能是否开启、注入故障概率、异常注入开关、异常故障生效概率决定是否触发，触发故障为对应模块随机一种异常。

表 5-17 异常故障

模块	故障名称	描述
全模块	SocketErr	网络连接异常。
	IORWErr	IO异常-读写关闭。
	IOUErr	IO异常-意外结束。
	NilPointerErr	空指针异常。
mysql	SQLErr	sql最顶层异常。
	SQLTimeoutErr	sql执行超时异常。
redis	RedisCommandUKErr	未知指令异常。
	RedisCommandArgErr	指令参数异常。

6 MAS-SDK 版本变更记录下载

MAS-SDK 版本变更记录及下载地址如下：

表 6-1 MAS-DB-SDK 版本变更

开发语言	版本及下载地址	变更描述
Java	v 1.2.0-RELEASE	初次发布。
Java	v 1.2.1-RELEASE	支持本地读单边写特性。
Java	v 1.2.2-RELEASE	支持强制路由source设置为ACTIVE。
Java	v 1.2.4-RELEASE	修复etcd故障后不能再watch key变化问题。
Java	v 1.2.6-RELEASE	增加连接池日志打印。
Java	v 1.2.7-RELEASE	修改select for update类型sql路由，改为路由到主库。

表 6-2 MAS-Redis-SDK 版本变更

开发语言	版本及下载地址	变更描述
Java	v 1.2.0-RELEASE	初次发布。
Java	v 1.2.2-RELEASE	增加byte系列方法，并重写对接spring-redis相关接口，支持script相关命令，优化etcd异常日志。

表 6-3 MAS-Mongo-SDK 版本变更

开发语言	版本及下载地址	变更描述
Java	v 1.2.0-RELEASE	初次发布。
Java	v 1.2.1-RELEASE	修复一些问题。
Java	v 1.2.2-RELEASE	修复一些问题。

表 6-4 MAS-GO-SDK 版本变更

开发语言	版本及下载地址	变更描述
GO	v 1.2.0-RELEASE	初次发布。