**Redis内存数据库研究**

目录

[一、 Redis简介 1](#_Toc472070629)

[二、 redis安装与启动 1](#_Toc472070630)

[三、 spring + redis的整合 2](#_Toc472070631)

[1、 本文使用maven下载依赖包： 2](#_Toc472070632)

[2、 Spring的applicationContext.xml加上redis的配置 3](#_Toc472070633)

[3、 redis.properties配置： 4](#_Toc472070634)

[4、 RedisDataSource接口定义四个方法： 4](#_Toc472070635)

[5、 RedisDataSource的实现： 5](#_Toc472070636)

[6、 第二层的封装：RedisClientTemplate 6](#_Toc472070637)

[7、 写一个方法测试代码： 55](#_Toc472070638)

[四、 对企业税务风险分析平台的TXT文本的json数据加载存储和获取的介绍 55](#_Toc472070639)

[1、 对文件进行解析成redis能存储格式的工具类 55](#_Toc472070640)

[2、 下面是工具类具体实现： 56](#_Toc472070641)

[3、 测试代码，输入文件 57](#_Toc472070642)

[4、 测试从数据库取出一条数据，循环一百万次的花费时间 57](#_Toc472070643)

## Redis简介

　　Redis是典型的NoSQL数据库服务器，它可以作为服务程序独立运行于自己的服务器主机。它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。

　　Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务 器。

在解决了很多通用性问题的同时，也为一些个性化问题提供了相关的解决方案，如索引引擎、统计排名、消息队列服务等。

## redis安装与启动

　　Redis是一个开源的使用ANSI?C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

　　官方下载地址：http://redis.io/download，不过官方没有64位的Windows下的可执行程序，目前有个开源的托管在github上, 地址：https://github.com/ServiceStack/redis-windows?点击这个地方，下载文件并解压到自己的电脑目录下。

　　本次研究使用：redis64-2.8.19.zip版本

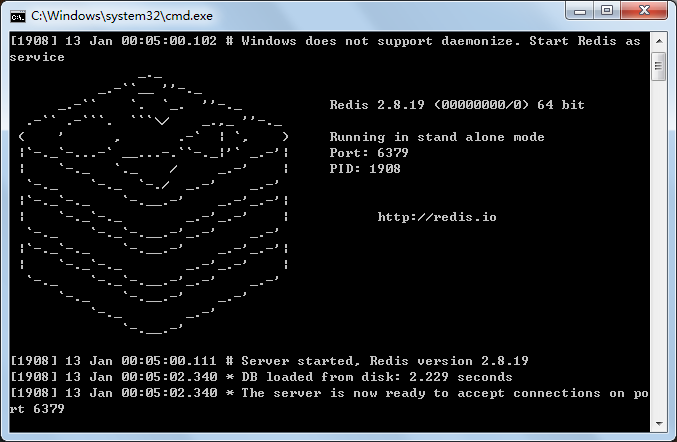
　　解压后目录如下：

|  |  |
| --- | --- |
| **文件名** | **简要** |
| redis-benchmark.exe | 基准测试（Redis性能测试工具） |
| redis-check-aof.exe | aof（更新日志检查） |
| redischeck-dump.exe | dump（检查本地数据文件） |
| redis-cli.exe | 客户端 |
| redis-server.exe | 服务器 |
| redis.windows.conf | 配置文件 |

启动redis数据库服务的命令：redis-server redis.windows.conf

可以将其保存为文件：startup.bat；下次就可以直接启动了。

启动成功后页面：



## spring + redis的整合

### 本文使用maven下载依赖包：

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.5.0.RELEASE</version>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

</dependency>

PS: 不同的redis.clients版本配置参数会有一点差别。

　　本文没有使用spring-data-redis进行整合，可以不加入spring-data-redis的jar包。

### Spring的applicationContext.xml加上redis的配置

**redis在applicationContext.xml上的完整配置：**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.foresee.db.\*\*"></context:component-scan>

<!--在return给pool时，是否提前进行validate操作-->

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

<value>classpath:redis.properties</value>

<!--要是有多个配置文件，只需在这里继续添加即可 -->

</list>

</property>

</bean>

<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<!--最大连接数, 默认8个-->

<property name="maxTotal" value="${redis.maxTotal}" />

<!--最大空闲连接数, 默认8个-->

<property name="maxIdle" value="${redis.maxIdle}" />

<!--获取连接时的最大等待毫秒数(值小于零:阻塞不确定的时间, 默认-1-->

<property name="maxWaitMillis" value="${redis.maxWaitMillis}" />

<!--在获取连接的时候检查有效性-->

<property name="testOnBorrow" value="${redis.testOnBorrow}"/>

<!--在return给pool时，是否提前进行validate操作-->

<property name="testOnReturn" value="${redis.testOnReturn}"/>

<!-- <property name="testWhileIdle" value="true"/> -->

</bean>

<!--ShardedJedisPool:多机分布式+连接池方式,是分片形式，可以连接有主备的redis服务端-->

<bean id="shardedJedisPool" class="redis.clients.jedis.ShardedJedisPool" scope="singleton">

<constructor-arg index="0" ref="jedisPoolConfig" />

<constructor-arg index="1">

<list>

<bean class="redis.clients.jedis.JedisShardInfo">

<constructor-arg name="host" value="${redis.host}" />

<constructor-arg name="port" value="${redis.port}" />

<constructor-arg name="timeout" value="${redis.timeout}" />

<!--没有password的构造方法，password为空时，不需要配置-->

<property name="password" value="${redis.password}" />

</bean>

</list>

</constructor-arg>

</bean>

<!--jedisPool:单机连接池方式,是非分片形式-->

<bean id="jedisPool" class="redis.clients.jedis.JedisPool" scope="singleton">

<constructor-arg index="0" ref="jedisPoolConfig" />

<constructor-arg index="1" value="${redis.host}" />

<constructor-arg index="2" value="${redis.port}" />

<constructor-arg index="3" value="${redis.timeout}" />

<constructor-arg index="4" value="${redis.password}" />

</bean>

</beans>

注意：配置了一个ShardedJedisPool，和一个JedisPool。这两个的区别:一个是分片形式，可以连接有主备的redis服务端，一个是单机的；这里全部展示出来，正式环境只需要配一个ShardedJedisPool，jedisPool一般用于单机测试。下面关于关于从jedisPool连接池获取连接的代码就不再展现，基本与ShardedJedisPool相同。ShardedJedisPool的集群配置通过在<list>标签内继续配置不同的<bean class="redis.clients.jedis.JedisShardInfo">

### redis.properties配置：

#redis中心

redis.host=127.0.0.1

redis.port=6379

redis.password=123456

redis.maxIdle=100

redis.maxTotal=300

redis.maxWaitMillis=1000

redis.testOnBorrow=true

redis.testOnReturn=true

redis.timeout=100000

redis.weight=1

### RedisDataSource接口定义四个方法：

**public** **interface** RedisDataSource {

**public** **abstract** ShardedJedis getShardedRedisClient();

**public** **void** returnResource(ShardedJedis shardedJedis);

**public** **void** returnResource(ShardedJedis shardedJedis,**boolean** broken);

**public** **void** shardedJedisDestroy();

}

这个接口主要是为了从redis的ShardedJedisPool连接池获取连接和将连接资源返还给连接池。

### RedisDataSource的实现：

首先注入配置好的ShardedJedisPool，这三个方法的作用：

getShardedRedisClient()：取得redis的客户端连接，可以执行命令了。

returnResource(ShardedJedis shardedJedis)：将资源返还给pool。

returnResource(ShardedJedis shardedJedis,**boolean** broken)： 出现异常后，将资源返还给pool。

shardedJedisDestroy()：应用关闭时，关闭连接池。

下面给出具体实现方法：

@Repository("redisDataSource")

**public** **class** RedisDataSourceImpl **implements** RedisDataSource {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(RedisDataSourceImpl.**class**);

@Autowired

**private** ShardedJedisPool shardedJedisPool;

/\*\*

\* 取得redis的客户端连接，采用了同步获取资源的方式，当高并发需要这个更安全

\*/

@Override

**public** **synchronized** ShardedJedis getShardedRedisClient() {

**try** {

ShardedJedis shardJedis = shardedJedisPool.getResource();

**return** shardJedis;

} **catch** (Exception e) {

***log***.error("getRedisClent error", e);

}

**return** **null**;

}

/\*\*

\* 将资源返还给pool

\*/

@Override

**public** **void** returnResource(ShardedJedis shardedJedis) {

shardedJedisPool.~~returnResource~~(shardedJedis);

}

/\*\*

\* 出现异常后，将资源返还给pool

\*/

@Override

**public** **void** returnResource(ShardedJedis shardedJedis, **boolean** broken) {

**if** (broken) {

shardedJedisPool.~~returnBrokenResource~~(shardedJedis);

} **else** {

shardedJedisPool.~~returnResource~~(shardedJedis);

}

}

/\*\*

\* 应用关闭时，关闭连接池

\*/

@Override

**public** **void** shardedJedisDestroy() {

shardedJedisPool.destroy();

}

}

### 第二层的封装：RedisClientTemplate

**基本实现对数据库进行增删改查等的所有操作，是自己实现的工具类。**

**代码就是映射性质的又一次调用jedis的方法而已，用了个broken来做标示符，决定返还资源的方式。**

**这一层的目的主要也是让再上层的调用不需要关心pool中链接的取得和返还问题了。**

@Repository("shardRedis")

**public** **class** ShardRedisClientTemplate {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(ShardRedisClientTemplate.**class**);

@Autowired

**private** RedisDataSource redisDataSource;

**public** **void** disconnect() {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

shardedJedis.disconnect();

}

/\*\*

\* 设置单个值

\*

\* **@param** key

\* **@param** value

\* **@return**

\*/

**public** String set(String key, String value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.set(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

/\*\*

\* 获取单个值

\*

\* **@param** key

\* **@return**

\*/

**public** String get(String key) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.get(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

//确认一个key是否存在

**public** Boolean exists(String key) {

Boolean result = **false**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.exists(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

//返回值的类型

**public** String type(String key) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.type(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

/\*\*

\* 设定一个key的活动时间（s）

\*/

**public** Long expire(String key, **int** seconds) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.expire(key, seconds);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

/\*\*

\* 在某个时间点失效

\*/

**public** Long expireAt(String key, **long** unixTime) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.expireAt(key, unixTime);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

//获得一个key的活动时间

**public** Long ttl(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.ttl(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

//

**public** **boolean** setbit(String key, **long** offset, **boolean** value) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**boolean** result = **false**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setbit(key, offset, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **boolean** getbit(String key, **long** offset) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**boolean** result = **false**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getbit(key, offset);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **long** setrange(String key, **long** offset, String value) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**long** result = 0;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setrange(key, offset, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String getrange(String key, **long** startOffset, **long** endOffset) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

String result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getrange(key, startOffset, endOffset);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String getSet(String key, String value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getSet(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long setnx(String key, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setnx(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String setex(String key, **int** seconds, String value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setex(key, seconds, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long decrBy(String key, **long** integer) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.decrBy(key, integer);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long decr(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.decr(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long incrBy(String key, **long** integer) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.incrBy(key, integer);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long incr(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.incr(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long append(String key, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.append(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String substr(String key, **int** start, **int** end) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.substr(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hset(String key, String field, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hset(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String hget(String key, String field) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hget(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hsetnx(String key, String field, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hsetnx(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String hmset(String key, Map<String, String> hash) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hmset(key, hash);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<String> hmget(String key, String... fields) {

List<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hmget(key, fields);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hincrBy(String key, String field, **long** value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hincrBy(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Boolean hexists(String key, String field) {

Boolean result = **false**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hexists(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long del(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.del(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hdel(String key, String field) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hdel(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hlen(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hlen(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> hkeys(String key) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hkeys(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<String> hvals(String key) {

List<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hvals(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Map<String, String> hgetAll(String key) {

Map<String, String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hgetAll(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

// ================list ====== l表示 list或 left, r表示right====================

**public** Long rpush(String key, String string) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.rpush(key, string);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long lpush(String key, String string) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lpush(key, string);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long llen(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.llen(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<String> lrange(String key, **long** start, **long** end) {

List<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String ltrim(String key, **long** start, **long** end) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.ltrim(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String lindex(String key, **long** index) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lindex(key, index);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String lset(String key, **long** index, String value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lset(key, index, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long lrem(String key, **long** count, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lrem(key, count, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String lpop(String key) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lpop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String rpop(String key) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.rpop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

//return 1 add a not exist value ,

//return 0 add a exist value

**public** Long sadd(String key, String member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sadd(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> smembers(String key) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.smembers(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long srem(String key, String member) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Long result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.srem(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String spop(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

String result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.spop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long scard(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Long result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.scard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Boolean sismember(String key, String member) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Boolean result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sismember(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String srandmember(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

String result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.srandmember(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zadd(String key, **double** score, String member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zadd(key, score, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrange(String key, **int** start, **int** end) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrem(String key, String member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrem(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Double zincrby(String key, **double** score, String member) {

Double result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zincrby(key, score, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrank(String key, String member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrank(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrevrank(String key, String member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrank(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrevrange(String key, **int** start, **int** end) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeWithScores(String key, **int** start, **int** end) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeWithScores(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeWithScores(String key, **int** start, **int** end) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeWithScores(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zcard(String key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zcard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Double zscore(String key, String member) {

Double result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zscore(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<String> sort(String key) {

List<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sort(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<String> sort(String key, SortingParams sortingParameters) {

List<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sort(key, sortingParameters);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zcount(String key, **double** min, **double** max) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zcount(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrangeByScore(String key, **double** min, **double** max) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScore(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrevrangeByScore(String key, **double** max, **double** min) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScore(key, max, min);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrangeByScore(String key, **double** min, **double** max, **int** offset, **int** count) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScore(key, min, max, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<String> zrevrangeByScore(String key, **double** max, **double** min, **int** offset, **int** count) {

Set<String> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScore(key, max, min, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeByScoreWithScores(String key, **double** min, **double** max) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScoreWithScores(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeByScoreWithScores(String key, **double** max, **double** min) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScoreWithScores(key, max, min);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeByScoreWithScores(String key, **double** min, **double** max, **int** offset, **int** count) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScoreWithScores(key, min, max, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeByScoreWithScores(String key, **double** max, **double** min, **int** offset, **int** count) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScoreWithScores(key, max, min, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zremrangeByRank(String key, **int** start, **int** end) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zremrangeByRank(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zremrangeByScore(String key, **double** start, **double** end) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zremrangeByScore(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long linsert(String key, LIST\_POSITION where, String pivot, String value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.linsert(key, where, pivot, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String set(**byte**[] key, **byte**[] value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.set(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] get(**byte**[] key) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.get(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Boolean exists(**byte**[] key) {

Boolean result = **false**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.exists(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String type(**byte**[] key) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.type(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long expire(**byte**[] key, **int** seconds) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.expire(key, seconds);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long expireAt(**byte**[] key, **long** unixTime) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.expireAt(key, unixTime);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long ttl(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.ttl(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] getSet(**byte**[] key, **byte**[] value) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getSet(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long setnx(**byte**[] key, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setnx(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String setex(**byte**[] key, **int** seconds, **byte**[] value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.setex(key, seconds, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long decrBy(**byte**[] key, **long** integer) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.decrBy(key, integer);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long decr(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.decr(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long incrBy(**byte**[] key, **long** integer) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.incrBy(key, integer);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long incr(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.incr(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long append(**byte**[] key, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.append(key, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] substr(**byte**[] key, **int** start, **int** end) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.substr(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hset(**byte**[] key, **byte**[] field, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hset(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] hget(**byte**[] key, **byte**[] field) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hget(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hsetnx(**byte**[] key, **byte**[] field, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hsetnx(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String hmset(**byte**[] key, Map<**byte**[], **byte**[]> hash) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hmset(key, hash);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<**byte**[]> hmget(**byte**[] key, **byte**[]... fields) {

List<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hmget(key, fields);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hincrBy(**byte**[] key, **byte**[] field, **long** value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hincrBy(key, field, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Boolean hexists(**byte**[] key, **byte**[] field) {

Boolean result = **false**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hexists(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hdel(**byte**[] key, **byte**[] field) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hdel(key, field);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long hlen(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hlen(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> hkeys(**byte**[] key) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hkeys(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Collection<**byte**[]> hvals(**byte**[] key) {

Collection<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hvals(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Map<**byte**[], **byte**[]> hgetAll(**byte**[] key) {

Map<**byte**[], **byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.hgetAll(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long rpush(**byte**[] key, **byte**[] string) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.rpush(key, string);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long lpush(**byte**[] key, **byte**[] string) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lpush(key, string);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long llen(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.llen(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<**byte**[]> lrange(**byte**[] key, **int** start, **int** end) {

List<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String ltrim(**byte**[] key, **int** start, **int** end) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.ltrim(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] lindex(**byte**[] key, **int** index) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lindex(key, index);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String lset(**byte**[] key, **int** index, **byte**[] value) {

String result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lset(key, index, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long lrem(**byte**[] key, **int** count, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lrem(key, count, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] lpop(**byte**[] key) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.lpop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] rpop(**byte**[] key) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.rpop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long sadd(**byte**[] key, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sadd(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> smembers(**byte**[] key) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.smembers(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long srem(**byte**[] key, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.srem(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] spop(**byte**[] key) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.spop(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long scard(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.scard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Boolean sismember(**byte**[] key, **byte**[] member) {

Boolean result = **false**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sismember(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** **byte**[] srandmember(**byte**[] key) {

**byte**[] result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.srandmember(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zadd(**byte**[] key, **double** score, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zadd(key, score, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrange(**byte**[] key, **int** start, **int** end) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrem(**byte**[] key, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrem(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Double zincrby(**byte**[] key, **double** score, **byte**[] member) {

Double result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zincrby(key, score, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrank(**byte**[] key, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrank(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zrevrank(**byte**[] key, **byte**[] member) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrank(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrevrange(**byte**[] key, **int** start, **int** end) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrange(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeWithScores(**byte**[] key, **int** start, **int** end) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeWithScores(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeWithScores(**byte**[] key, **int** start, **int** end) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeWithScores(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zcard(**byte**[] key) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zcard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Double zscore(**byte**[] key, **byte**[] member) {

Double result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zscore(key, member);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<**byte**[]> sort(**byte**[] key) {

List<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sort(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<**byte**[]> sort(**byte**[] key, SortingParams sortingParameters) {

List<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.sort(key, sortingParameters);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zcount(**byte**[] key, **double** min, **double** max) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zcount(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrangeByScore(**byte**[] key, **double** min, **double** max) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScore(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrangeByScore(**byte**[] key, **double** min, **double** max, **int** offset, **int** count) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScore(key, min, max, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeByScoreWithScores(**byte**[] key, **double** min, **double** max) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScoreWithScores(key, min, max);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrangeByScoreWithScores(**byte**[] key, **double** min, **double** max, **int** offset, **int** count) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrangeByScoreWithScores(key, min, max, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrevrangeByScore(**byte**[] key, **double** max, **double** min) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScore(key, max, min);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<**byte**[]> zrevrangeByScore(**byte**[] key, **double** max, **double** min, **int** offset, **int** count) {

Set<**byte**[]> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScore(key, max, min, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeByScoreWithScores(**byte**[] key, **double** max, **double** min) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScoreWithScores(key, max, min);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Set<Tuple> zrevrangeByScoreWithScores(**byte**[] key, **double** max, **double** min, **int** offset, **int** count) {

Set<Tuple> result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zrevrangeByScoreWithScores(key, max, min, offset, count);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zremrangeByRank(**byte**[] key, **int** start, **int** end) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zremrangeByRank(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long zremrangeByScore(**byte**[] key, **double** start, **double** end) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.zremrangeByScore(key, start, end);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Long linsert(**byte**[] key, LIST\_POSITION where, **byte**[] pivot, **byte**[] value) {

Long result = **null**;

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.linsert(key, where, pivot, value);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** List<Object> pipelined(ShardedJedisPipeline shardedJedisPipeline) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

List<Object> result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.~~pipelined~~(shardedJedisPipeline);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Jedis getShard(**byte**[] key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Jedis result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getShard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Jedis getShard(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Jedis result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getShard(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** JedisShardInfo getShardInfo(**byte**[] key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

JedisShardInfo result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getShardInfo(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** JedisShardInfo getShardInfo(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

JedisShardInfo result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getShardInfo(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** String getKeyTag(String key) {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

String result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getKeyTag(key);

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Collection<JedisShardInfo> getAllShardInfo() {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Collection<JedisShardInfo> result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getAllShardInfo();

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

**public** Collection<Jedis> getAllShards() {

ShardedJedis shardedJedis = redisDataSource.getShardedRedisClient();

Collection<Jedis> result = **null**;

**if** (shardedJedis == **null**) {

**return** result;

}

**boolean** broken = **false**;

**try** {

result = shardedJedis.getAllShards();

} **catch** (Exception e) {

***log***.error(e.getMessage(), e);

broken = **true**;

} **finally** {

redisDataSource.returnResource(shardedJedis, broken);

}

**return** result;

}

}

### 写一个方法测试代码：

**public** **static** **void** main(String[] args) {

ApplicationContext ac = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

ShardRedisClientTemplate shardRedis = (ShardRedisClientTemplate) ac.getBean("shardRedis");

shardRedis.set("a", "abc");

System.out.println(shardRedis.get("a"));

}

**到这里redis的基本开发环境搭建完成。**

## 对企业税务风险分析平台的TXT文本的json数据加载存储和获取的介绍

### 对文件进行解析成redis能存储格式的工具类

TXT文件存储的json格式如下：

{

"DWBM":"1000100002",

"SWMC":"某某公司门店的月收入情况",

"SJ":"2008\/12\/1",

"YYE":1001000.00,

"SE":10100.00,

"SF":null

},

{

"DWBM":"1000100002",

"SWMC":"某某公司门店的月收入情况",

"SJ":"2016\/1\/1",

"YYE":1001000.00,

"SE":10100.00,

"SF":null

},

{

"DWBM":"1000100002",

"SWMC":"某某公司门店的月收入情况",

"SJ":"2010\/11\/1",

"YYE":1001000.00,

"SE":10100.00,

"SF":null

}

对文件进行解析，每一个json按照K-V模式存储，K为DWBM的值，V为redis的map，该map的K-V中K存储为SJ字符串格式（“YYYY/MM”），V存储整个的json的值（注意：redis的map的K-V都必须是字符串）。

### 下面是工具类具体实现：

@Service("readFileOperate")

**public** **class** ReadFileOperate {

@Autowired

**private** ShardRedisClientTemplate shardRedis;

**public** **void** readFileStream(File file){

**try**{

FileInputStream fileInputStream = **new** FileInputStream(file);

BufferedInputStream bufferedInputStream = **new** BufferedInputStream(fileInputStream);

InputStreamReader inputStreamReader = **new** InputStreamReader(bufferedInputStream,"utf-8");

BufferedReader input = **new** BufferedReader(inputStreamReader, (**int**) file.length());

String str = "", app = "" , sj;

**while**((str = input.readLine()) != **null**){

app=app+str;

**if** (str.indexOf("}") == 0) {

app = app.replace("},", "}");

JSONObject object = JSONObject.*fromObject*(app);

sj = object.get("SJ").toString();

shardRedis.hset(object.get("DWBM").toString(), sj.substring(0,sj.lastIndexOf("/")), app);

app="";

}

}

}**catch**(Exception e){

e.printStackTrace();

}

}

}

### 测试代码，输入文件

**public** **static** **void** main(String[] args) {

ApplicationContext ac = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

String path = "D:\\zgj\\file\\dwss.json";

// String path1 = "D:\\TEXT.txt";

File file = **new** File(path);

ReadFileOperate readFileOperate = (ReadFileOperate) ac.getBean("readFileOperate");

**long** start = System.*currentTimeMillis*();

readFileOperate.readFileStream(file);

//存储83万条数据入redis，平均花费170秒

**long** end = System.*currentTimeMillis*();

System.***out***.println((end-start)+"毫秒");

}

### 测试从数据库取出一条数据，循环一百万次的花费时间

**public** **static** **void** main(String[] args) {

ApplicationContext ac = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

ShardRedisClientTemplate shardRedis = (ShardRedisClientTemplate) ac.getBean("shardRedis");

**long** start = System.*currentTimeMillis*();

**for**(**int** i=0;i<10000;i++){

//循环了一百万次，花费220698毫秒

String str = shardRedis.hget("1000100002", "2014/9");

**if**(str !=**null**){

JSONObject object = JSONObject.*fromObject*(str);

**double** yye = (**double**) object.get("YYE");

**double** se = (**double**) object.get("SE");

System.***out***.println(yye/se);

}

}

shardRedis.hdel("1000100004","2014/10");//删除K为"1000100003"的map里面K为"2014/10"的数据

**long** del = shardRedis.del("1000100003");//删除K为"1000100003"的所有数据

**long** end = System.*currentTimeMillis*();

System.***out***.println((end-start)+"毫秒" +del);

}