Apusic 金蝶天燕

生成時間：2026-10-23

開發手冊

金蝶Apusic分佈式緩存 V2.0

版權所有 © 深圳市金蝶天燕雲計算股份有限公司2026。保留所有權利。

版權聲明

本文件所涉及的軟件著作權、版權等知識產權已依法進行了註冊，由金蝶天燕雲計算股份有限公司合法擁有。受《中華人民共和國著作權法》《計算機軟件保護條例》《知識產權保護條例》和相關國際版權條約、法律、法規以及其它知識產權法律和條約的保護。未經授權許可，不得非法使用。

免責聲明

本文件包含的版權資訊由金蝶天燕雲計算股份有限公司合法擁有，受法律的保護，金蝶天燕雲計算股份有限公司對本文件可能涉及到的非金蝶天燕雲計算股份有限公司的資訊不承擔任何責任。在法律允許的範圍內，您可以查閱並僅能夠在《中華人民共和國著作權法》規定的合法範圍內複製和打印本文件。任何單位和個人未經金蝶天燕雲計算股份有限公司書面授權許可，不得使用、修改、再發佈本文件的任何部分和內容，否則將被視為侵權，金蝶天燕雲計算股份有限公司有依法追究其責任的權利。

本文件如有更新，不另行通知。對本文件中的問題您可向金蝶天燕雲計算股份有限公司告知或查詢。未經本公司明確授予的任何權利均予保留。

商標聲明

Apusic 是深圳市金蝶天燕雲計算股份有限公司向中華人民共和國國家商標局申請註冊的註冊商標，註冊商標專用權由金蝶天燕合法擁有，受法律保護。未經金蝶天燕的書面許可，任何單位及個人不得以任何方式或理由對該商標的任何部分進行使用、複製、修改、傳播、抄錄或與其它產品捆綁使用銷售。凡侵犯金蝶天燕商標權的，金蝶天燕將依法追究其法律責任。本文件提及的其他所有商標或註冊商標，由各自的所有人擁有。

目錄

- 1 前言
  - 1.1 適用對象
  - 1.2 相關文檔
  - 1.3 技術支持
- 2 客戶端應用開發
  - 2.1 Java應用開發
    - 2.1.1 Spring Boot項目集成
      - 2.1.1.1 創建項目並添加配置
      - 2.1.1.2 配置RedisTemplate
      - 2.1.1.3 使用示例
    - 2.1.2 Redisson 使用緩存示例
      - 2.1.2.1 快速開始
  - 2.2 Python應用開發
  - 2.3 Go應用開發
  - 2.4 KVDB操作命令

1 前言

本文檔為金蝶Apusic鍵值數據庫（KVDB）V2.0的開發使用說明，幫助用戶快速學習如何使用金蝶Apusic鍵值數據庫進行開發。

1.1 適用對象

本文檔適用於開發工程師、軟件架構師及研發經理等人員。

1.2 相關文檔

了解更多Apusic KVDB V2.0產品相關的信息，請參閱以下Apusic KVDB V2.0產品手冊文檔集：

序號	手冊文檔	說明
1	金蝶Apusic鍵值數據庫 V2.0 快速使用手冊	簡單介紹了如何快速上手使用KVDB。
2	金蝶Apusic鍵值數據庫 V2.0 安裝手冊	詳細介紹如何在各操作系統上安裝KVDB，以及KVDB服務啟停操作和產品註冊過程。
3	金蝶Apusic鍵值數據庫 V2.0 鍵值核心用戶手冊	詳細介紹 KVDB 相關功能的使用、配置、管理及配套工具的使用方法。
4	金蝶Apusic鍵值數據庫 V2.0 管控台用戶手冊	詳細介紹KVDB管控台相關功能的使用和操作說明。
5	金蝶Apusic鍵值數據庫 V2.0 開發手冊	詳細介紹基於各開發語言進行KVDB客戶端應用開發的說明。
6	金蝶Apusic鍵值數據庫 V2.0 遷移手冊	詳細介紹KVDB歷史版本遷移升級到V2.0版本的說明，以及Redis遷移到KVDB的說明。
7	金蝶Apusic鍵值數據庫 V2.0 運維手冊	詳細介紹KVDB的監控、運維、安全加固等運維說明。
8	金蝶Apusic鍵值數據庫 V2.0 性能優化手冊	詳細介紹KVDB性能調優的說明。

1.3 技術支持

KVDB產品提供全面的技術支持服務，您可以通過以下方式獲得技術支持：

- 網址：www.apusic.com
- 電話：400-855-5800
- 郵箱：support@apusic.com
- 金蝶雲社區：https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN

您在取得技術支持時，請提供如下信息：

1. 您的姓名
2. 公司信息與聯繫方式
3. 操作系統及其版本
4. 產品版本號
5. 出現異常及錯誤的日誌、截圖等詳細信息

2 客戶端應用開發

本文使用 Java、Python、Go 三種常用語言展示KVDB對各種語言Redis客戶端的兼容性。

2.1 Java應用開發

Java開發Redis客戶端應用推薦使用Jedis。Jedis 是Redis 官方推薦的、輕量級且高性能的同步 Java 客戶端，它以接近原生命令的方式讓 Java 開發者能夠方便、高效地連接、操作和管理Redis數據庫。

以下通過使用Jedis連接KVDB的單機、哨兵和集群模式部署的鍵值核心服務，並進行數據讀寫操作提供開發示例。

- 單節點連接與使用

```
Jedis kvdb = new Jedis("10.0.1.100 (示例IP, 請替換為實際IP) ", 6359);
System.out.println(kvdb.ping());
System.out.println(kvdb.set("key11", "value11"));
System.out.println(kvdb.get("key11"));
```

- 哨兵模式連接與使用

```
Set<String> sentinels = new HashSet<String>(Arrays.asList(
    "10.0.1.110 (示例IP, 請替換為實際IP) :27000",
    "10.0.1.110 (示例IP, 請替換為實際IP) :27001",
    "10.0.1.110 (示例IP, 請替換為實際IP) :27002"
));
JedisSentinelPool pool = new JedisSentinelPool("mymaster",
sentinels, jedisPoolConfig);
Jedis jedis = pool.getResource();
System.out.println(jedis.set("key12", "value12"));
System.out.println(jedis.get("key12"));
```

- 集群模式連接與使用

```
Set<HostAndPort> clusterNodes = new HashSet<>();
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7000));
```

```
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7001));
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7002));
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7003));
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7004));
clusterNodes.add(new HostAndPort("10.0.1.110 (示例IP, 請替換為實際IP) ",
7005));
JedisCluster ac = new JedisCluster(clusterNodes,
genericObjectPoolConfig);
System.out.println(ac.set("test1", "a"));
System.out.println(ac.set("1test", "b"));
System.out.println(ac.set("aaatest", "c"));
System.out.println("-------------------------");
System.out.println(ac.get("test1"));
System.out.println(ac.get("1test"));
System.out.println(ac.get("aaatest"));
```

2.1.1 Spring Boot項目集成

KVDB兼容Redis數據存儲協議，外部客戶端可無縫切換使用KVDB產品。

2.1.1.1 創建項目並添加配置

```
# application.yml
spring:
  redis:
    host: localhost        # Redis地址
    port: 6359             # 端口
    timeout: 2000ms        # 連接超時時間
    database: 0            # 使用0號數據庫 (0-15)

    # 連接池配置 (重要! )
    lettuce:
      pool:
```

```
        max-active: 16    # 最大連接數
        max-idle: 8       # 最大空閒連接
        min-idle: 4       # 最小空閒連接
        max-wait: 1000ms # 獲取連接最大等待時間
```

2.1.1.2 配置RedisTemplate

```
@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(
            RedisConnectionFactory factory) {

        RedisTemplate<String, Object> template = new RedisTemplate<>
();
        template.setConnectionFactory(factory);

        // 使用String序列化Key（重要！）
        template.setKeySerializer(new StringRedisSerializer());
        template.setHashKeySerializer(new StringRedisSerializer());

        // 使用JSON序列化Value
        template.setValueSerializer(new
Jackson2JsonRedisSerializer<>(Object.class));
        template.setHashValueSerializer(new
Jackson2JsonRedisSerializer<>(Object.class));

        return template;
    }
}
```

2.1.1.3 使用示例

下面是緩存參與審批流程狀態管理的一個實例。

```java
@Service
public class ApprovalService {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    /**
     * 存儲審批狀態
     */
    public void saveApprovalStatus(String approvalId,
                                   String status,
                                   String approver) {
        String key = "approval:status:" + approvalId;

        Map<String, String> data = new HashMap<>();
        data.put("status", status);
        data.put("approver", approver);
        data.put("updateTime", LocalDateTime.now().toString());
        data.put("expireAt", LocalDateTime.now()
            .plusDays(7).toString()); // 審批數據保留7天

        // 使用Hash存儲
        redisTemplate.opsForHash().putAll(key, data);

        // 設置過期時間
        redisTemplate.expire(key, 7, TimeUnit.DAYS);
    }

    /**
     * 獲取審批進度
     */
    public Map<Object, Object> getApprovalProgress(String
approvalId) {
        String key = "approval:status:" + approvalId;
        return redisTemplate.opsForHash().entries(key);
    }
```

```
    /**
     * 批量獲取待辦事項
     */
    public List<String> getPendingApprovals(String approver) {
        String key = "approval:pending:" + approver;
        List<Object> list = redisTemplate.opsForList().range(key, 0,
-1);

        return list != null ?
            list.stream()
                .map(Object::toString)
                .collect(Collectors.toList()) :
            Collections.emptyList();
    }
}
```

2.1.2 Redisson 使用緩存示例

Redisson 是Redis 的 Java 客戶端與實時數據平台，為開發者提供最方便、最易用的使用方式。通過 Redisson 對象，它在 Java 代碼與Redis 之間建立了優雅的抽象層，讓開發者專注於業務邏輯與數據模型。同時，Redisson 極大地擴展了Redis的功能，實現了大量原生所不具備的分佈式特性，包括分佈式集合、分佈式鎖、分佈式對象、分佈式服務等。

本節演示KVDB對Redisson框架的開發支持和兼容特性，提供相關代碼示例供參考。

2.1.2.1 快速開始

首先進行依賴配置

```
<!-- pom.xml -->
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson-spring-boot-starter</artifactId>
    <version>3.27.1</version>
</dependency>
```

```yaml
# application.yml - 單機模式
spring:
  redis:
    redisson:
      config: |
        singleServerConfig:
          address: "redis://127.0.0.1:6359"
          password: "your_password"
          database: 0
          # 連接池設置
          connectionPoolSize: 64
          connectionMinimumIdleSize: 24
          # 超時設置（毫秒）
          connectTimeout: 10000
          timeout: 3000
          retryAttempts: 3
        # 使用JSON序列化
        codec: !<org.redisson.codec.JsonJacksonCodec> {}
```

以下是在代碼中集成的不同模式的緩存的使用方式和部分參數的配置。

```java
@Configuration
public class RedissonConfig {

    @Bean(destroyMethod = "shutdown")
    public RedissonClient redissonClient() {
        Config config = new Config();

        // 單機模式
        config.useSingleServer()
              .setAddress("redis://127.0.0.1:6359")
              .setPassword("your_password")
              .setDatabase(0)
              .setConnectionPoolSize(64)
              .setConnectionMinimumIdleSize(10);
```

```
        // 集群模式（金融政務生產環境推薦）
        // config.useClusterServers()
        //        .addNodeAddress(
        //            "redis://node1:6359",
        //            "redis://node2:6359",
        //            "redis://node3:6359"
        //        )
        //        .setScanInterval(2000); // 集群狀態掃描間隔

        // 哨兵模式
        // config.useSentinelServers()
        //        .setMasterName("mymaster")
        //        .addSentinelAddress(
        //            "redis://sentinel1:26359",
        //            "redis://sentinel2:26359"
        //        );

        // 序列化配置
        config.setCodec(new JsonJacksonCodec());

        // 線程配置
        config.setThreads(16);  // 處理Redis響應的線程數
        config.setNettyThreads(32); // Netty I/O線程數

        return Redisson.create(config);
    }
}
```

以下同樣使用審批流示例：

```
@Service
public class ApprovalWorkflowService {

    @Autowired
    private RedissonClient redisson;
```

```
    /**
     * 分佈式審批流程
     */
    public void processApproval(String applicationId,
                                String approver,
                                boolean approved) {

        // 1. 獲取流程鎖
        RLock flowLock = redisson.getLock("approval:flow:" +
applicationId);

        if (flowLock.tryLock()) {
            try {
                // 2. 獲取當前流程狀態
                RMap<String, String> flowState =
                    redisson.getMap("approval:state:" +
applicationId);

                String currentStep = flowState.get("currentStep");
                String status = flowState.get("status");

                // 3. 檢查是否可以審批
                if (!"PENDING".equals(status)) {
                    throw new IllegalStateException("申請已處理完成");
                }

                // 4. 更新審批狀態
                flowState.put("approver", approver);
                flowState.put("approved", String.valueOf(approved));
                flowState.put("approvalTime",
LocalDateTime.now().toString());
                flowState.put("status", approved ? "APPROVED" :
"REJECTED");

                // 5. 發佈審批完成事件
```

```
                RTopic topic =
redisson.getTopic("approval:completed");
                topic.publishAsync(new ApprovalEvent(applicationId,
approved));

                // 6. 記錄審批歷史
                RList<ApprovalRecord> history =
                    redisson.getList("approval:history:" +
applicationId);
                history.add(new ApprovalRecord(approver, approved,
LocalDateTime.now()));

            } finally {
                flowLock.unlock();
            }
        }
    }

    /**
     * 多級會簽（需要所有審批人同意）
     */
    public boolean multiSignApproval(String applicationId,
List<String> approvers) {
        String latchKey = "approval:latch:" + applicationId;
        RCountDownLatch latch =
redisson.getCountDownLatch(latchKey);

        // 初始化計數器
        latch.trySetCount(approvers.size());

        // 每個審批人獨立處理
        for (String approver : approvers) {
            // 異步提交給審批人
            submitToApprover(applicationId, approver, latchKey);
        }
```

```
        try {
            // 等待所有審批人完成（最多1小時）
            return latch.await(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return false;
        }
    }
}
```

2.2 Python應用開發

- 單節點連接與使用

```
import redis
import time

r = redis.Redis(host='10.0.1.100（示例IP，請替換為實際IP）', port=6359,
db=0, decode_responses=True)
print(r.set("key1", "value1"))
print(r.get("key1"))
r.expire("key1", 10)
print(r.ttl("key1"))
time.sleep(1)
print(r.ttl("key1"))
```

- 哨兵模式連接與使用

```
from redis.sentinel import Sentinel

SENTINELADDRS = [("10.0.1.120（示例IP，請替換為實際IP）", "26359"),
("10.0.1.120（示例IP，請替換為實際IP）", "26360"), ("10.0.1.120（示例IP，
請替換為實際IP）", "26361")]
def sentinel():
    kvdb = redis.Sentinel(sentinels=SENTINELADDRS)
```

```
    # 獲取主實例
    master = kvdb.master_for("mymaster")
    # 獲取從實例
    slave = kvdb.slave_for("mymaster")
    # 主實例中寫入
    print(master.set("key2", "value2"))
    time.sleep(2)
    # 從實例中獲取
    print(slave.get("key2"))
```

- 集群模式連接與使用

```
from redis.cluster import RedisCluster

CLUSTERADDRS = [{"host":"10.0.1.120 (示例IP, 請替換為實際IP) ",
"port":"6359"}, {"host":"10.0.1.120 (示例IP, 請替換為實際IP) ",
"port":"6360"}, {"host":"10.0.1.120 (示例IP, 請替換為實際IP) ",
"port":"6361"}]
def cluster():

kvdb_cluster=RedisCluster(startup_nodes=CLUSTERADDRS,decode_responses=T
   print(kvdb_cluster.set("key3","value3"))
   print(kvdb_cluster.get("key3"))
```

2.3 Go應用開發

- 單節點連接與使用

```
package main

import (
    "fmt"
    "github.com/go-redis/redis"
)
```

```
func main() {
    rdb := redis.NewClient(&redis.Options{
        Addr: "10.0.1.120 (示例IP, 請替換為實際IP) :6359",
    })
    if err := rdb.Ping().Err(); err != nil {
        fmt.Println("Ping error:", err)
    } else {
        fmt.Println("Ping:", rdb.Ping().Val())
    }
}
```

- 哨兵模式連接與使用

```
package main

import (
    "fmt"
    "github.com/go-redis/redis/v8"
)

func main() {
    rdb := redis.NewFailoverClient(&redis.FailoverOptions{
        MasterName: "mymaster",
        SentinelAddrs: []string{"10.0.1.120 (示例IP, 請替換為實際
IP) :26359", "10.0.1.120 (示例IP, 請替換為實際IP) :26360", "10.0.1.120
 (示例IP, 請替換為實際IP) :26361"},
    })
    defer rdb.Close()
    _, err := rdb.Set("hahakey", "hahavalue", 0).Result()
    if err != nil {
        fmt.Println("Set error:", err)
    } else {
        fmt.Println("Set hahakey: hahavalue")
    }
}
```

- 集群模式連接與使用

```
package main

import (
    "fmt"
    "github.com/go-redis/redis/v8"
)

func main() {
    addrs := []string{"10.0.1.120（示例IP，請替換為實際IP）:6359", "10.0.1.120（示例IP，請替換為實際IP）:6360", "10.0.1.120（示例IP，請替換為實際IP）:6361"}
    rdb := redis.NewClusterClient(&redis.ClusterOptions{
        Addrs: addrs,
    })
    if err := rdb.Ping().Err(); err != nil {
        fmt.Println("Ping error:", err)
    } else {
        fmt.Println("Ping:", rdb.Ping().Val())
    }
}
```

2.4 KVDB操作命令

命令	說明
ACL LOAD	從配置的ACL文件中重新加載ACL
ACL SAVE	在已配置的ACL文件中保存當前的ACL規則
ACL LIST	列出ACL配置文件格式的當前ACL規則
ACL USERS	列出所有已配置的ACL規則的用戶名
ACL GETUSER username	獲取特定ACL用戶的規則
ACL SETUSER username [rule [rule ...]]	修改或創建特定ACL用戶的規則

ACL DELUSER username [username ...]	刪除指定的ACL用戶和相關規則
ACL CAT [categoryname]	列出類別內的ACL類別或命令
ACL GENPASS [bits]	為ACL用戶生成偽隨機安全密碼
ACL WHOAMI	返回與當前連接關聯的用戶的名稱
ACL LOG [count or RESET]	列出最新的ACL安全事件
ACL HELP	顯示有關不同的子命令的有用文本
APPEND key value	將值附加到鍵
ASKING	在 ASK 重定向後由群集客戶端發送
AUTH [username] password	對服務器進行身份驗證
BGREWRITEAOF	異步重寫僅附加文件
BGSAVE [SCHEDULE]	異步將數據集保存到磁盤
BITCOUNT key [start end [BYTE|BIT]]	計算字符串中的設置位
BITFIELD key [GET encoding offset] [SET encoding offset value] [INCRBY encoding offset increment] [OVERFLOW WRAP|SAT|FAIL]	在字符串上執行任意位字段整數操作
BITFIELD_RO key GET encoding offset	在字符串上執行任意位字段整數操作，禁用的只讀變體
BITOP operation destkey key [key ...]	在字符串之間執行按位操作
BITPOS key bit [start [end [BYTE|BIT]]]	在字符串中找到第一個位設置或清除
BLPOP key [key ...] timeout	阻塞式刪除並返回第一個元素
BRPOP key [key ...] timeout	阻塞式刪除並返回最後一個元素
BRPOPLPUSH source target timeout	從列表中取出最後一個元素，並插入到另外一個列表的頭部； 如果列表沒有元素會阻塞列表直到等待超時或發現可彈出元素為止。
BLMOVE source destination FROM-TOP|FROM-BOTTOM TO-TOP| TO-BOTTOM timeout	阻塞式返回並刪除存儲在源列表的第一個或最後一個元素（頭尾取決於wherefrom參數），並將該元素壓入到存儲目標列表的第一個或最後一個元素（頭尾取決於whereto參數）

LMPOP numkeys key [key ...] LEFT|RIGHT [COUNT count]	從提供的鍵名列表中彈出第一個非空列表鍵中的一個或多個元素
BLMPOP timeout numkeys key [key ...] LEFT|RIGHT [COUNT count]	阻塞式從提供的鍵名列表中彈出第一個非空列表鍵中的一個或多個元素；或阻塞連接直到另一個客戶端推送到它或直到超時
BZPOPMIN key [key ...] timeout	阻塞式刪除並將成員從一個或多個有序集中返回最低分數，或阻塞連接直到另一個客戶端推送到它或直到超時
BZPOPMAX key [key ...] timeout	阻塞式刪除並將成員從一個或多個有序集中返回最高分數，或阻塞連接直到另一個客戶端推送到它或直到超時
CLIENT CACHING YES|NO	指示服務器在下一個請求中跟踪或不鍵
CLIENT ID	返回當前連接的客戶端ID
CLIENT INFO	返回有關當前客戶端連接的信息
CLIENT KILL [ip:port] [ID client-id] [TYPE normal|master|slave|pubsub] [USER username] [ADDR ip:port] [LADDR ip:port] [SKIPME yes/no]	殺死客戶端的連接
CLIENT LIST [TYPE normal|master|replica|pubsub] [ID client-id [client-id ...]]	獲取客戶端連接列表
CLIENT GETNAME	獲取當前的連接名稱
CLIENT GETREDIR	獲取跟踪通知重定向客戶端ID（如果有）
CLIENT UNPAUSE	恢復暫停的客戶的處理
CLIENT PAUSE timeout [WRITE|ALL]	停止客戶端的處理命令一段時間
CLIENT REPLY ON|OFF|SKIP	指示服務器是否回覆命令
CLIENT SETNAME connection-name	設置當前連接名稱
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix [PREFIX prefix ...]] [BCAST] [OPTIN] [OPTOUT] [NOLOOP]	啟用或禁用服務器輔助客戶端緩存支持
CLIENT TRACKINGINFO	返回關於當前連接的服務器輔助客戶端緩存的信息
CLIENT UNBLOCK client-id [TIMEOUT|ERROR]	取消阻止從不同連接中被阻塞命令阻塞的客戶端

COMMAND	獲取 Redis 命令詳細信息數組
COMMAND COUNT	獲取Redis命令的總數
COMMAND GETKEYS	給定完整的 Redis 命令提取鍵
COMMAND INFO command-name [command-name ...]	獲取特定Redis命令詳細信息的數組
CONFIG GET parameter [parameter ...]	獲取配置參數的值
CONFIG REWRITE	用內存配置重寫配置文件
CONFIG SET parameter value [parameter value ...]	將配置參數設置為給定的值
CONFIG RESETSTAT	重置 INFO 返回的統計信息
COPY source destination [DB destination-db] [REPLACE]	複製一個鍵
DBSIZE	返回所選數據庫中的鍵數
DEBUG OBJECT key	獲取有關鍵的調試信息
DEBUG SEGFAULT	使服務器崩潰
DECR key	一個鍵的整數值減一
DECRBY key decrement	按給定的數字減少一個鍵的整數值
DEL key [key ...]	刪除一個鍵
DISCARD	放棄在多次執行後發出的所有命令
DUMP key	返回存儲在指定鍵中的值的序列化版本
DELEX key	根據值的比較結果（值相等/不等、哈希摘要匹配/不匹配）來條件性刪除一個 Key
DIGEST key	計算key對應值的摘要（SM3/SHA256等）
ECHO message	回顯給定的字符串
EVAL script numkeys [key [key ...]] [arg [arg ...]]	執行一個Lua腳本服務器端
EVALSHA sha1 numkeys [key [key ...]] [arg [arg ...]]	執行 Lua 腳本服務器端
EXEC	執行 MULTI
EXISTS key [key ...]	判斷key是否存在

EXPIRE key seconds [NX	XX
EXPIREAT key timestamp [NX	XX
EXPIRETIME key	獲取鍵的到期 Unix 時間戳
FAILOVER [TO host port [FORCE]] [ABORT] [TIMEOUT milliseconds]	在此服務器與其副本之一之間啟動協調故障轉移
FLUSHALL [ASYNC	SYNC]
FLUSHDB [ASYNC	SYNC]
FUNCTION LOAD	加載服務器端函數（替代Lua腳本）
FUNCTION DELETE	刪除指定函數
FUNCTION FLUSH	清空所有函數
FUNCTION LIST	列出所有函數
FUNCTION STATS	查看函數執行統計
GEOADD key [NX	XX] [CH] longitude latitude member [longitude latitude member ...]
GEOHASH key member [member ...]	將地理空間索引的成員作為標準 geohash 字符串返回
GEOPOS key member [member ...]	返回地理空間索引成員的經度和緯度
GEODIST key member1 member2 [m	km
GEORADIUS key longitude latitude radius m	km
GEORADIUSBYMEMBER key member radius m	km
GEOSEARCH key [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m	km
GEOSEARCHSTORE destination source [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m	km
GET key	獲取一個鍵的值
GETBIT key offset	返回存儲在 key

GETDEL key	的字符串值中偏移量處的位值，獲取某個key的值並刪除key
GETRANGE key start end	獲取存儲在鍵中的字符串的子字符串
GETSET key value	設置一個鍵的字符串值並返回它的舊值
HDEL key field [field ...]	刪除一個或多個哈希字段
HELLO [protover [AUTH username password] [SETNAME clientname]]	與 Redis 握手
HEXISTS key field	判斷一個hash字段是否存在
HGET key field	獲取哈希字段的值
HGETALL key	獲取哈希中的所有字段和值
HINCRBY key field increment	將散列字段的整數值增加給定的數字
HINCRBYFLOAT key field increment	將哈希字段的浮點值增加給定的數量
HKEYS key	獲取散列中的所有字段
HLEN key	獲取哈希中的字段數
HMGET key field [field ...]	獲取所有給定哈希字段的值
HMSET key field value [field value ...]	將多個哈希字段設置為多個值
HSET key field value [field value ...]	設置一個哈希字段的字符串值
HSETNX key field value	設置一個哈希字段的值，僅當該字段不存在時
HRANDFIELD key [count [WITHVALUES]]	從散列中獲取一個或多個隨機字段
HSTRLEN key field	獲取哈希字段值的長度
HVALS key	獲取散列中的所有值
HOTKEYS [[value] [opt] ...]	檢測並列出當前熱鍵
INCR key	鍵的整數值加一
INCRBY key increment	將鍵的整數值增加給定的數量
INCRBYFLOAT key increment	將鍵的浮點值增加給定的數量
INFO [section]	獲取有關服務器的信息和統計信息

LOLWUT [VERSION version]	展示一些計算機藝術和 Redis 版本
KEYS pattern	找到所有匹配給定模式的鍵
LASTSAVE	獲取上次成功保存到磁盤的 UNIX 時間戳
LINDEX key index	通過索引從列表中獲取元素
LINSERT key BEFORE	AFTER pivot element
LLEN key	獲取列表的長度
LPOP key [count]	刪除並獲取列表中的第一個元素
LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]	返回列表中匹配元素的索引
LPUSH key element [element ...]	將一個或多個元素添加到列表中
LPUSHX key element [element ...]	將元素添加到列表中，僅當列表存在時
LRANGE key start stop	從列表中獲取一系列元素
LREM key count element	從列表中刪除元素
LSET key index element	通過索引設置列表中元素的值
LTRIM key start stop	將列表修剪到指定範圍
MEMORY DOCTOR	輸出內存問題報告
MEMORY HELP	顯示有關不同子命令的有用文本
MEMORY MALLOC-STATS	顯示分配器內部統計信息
MEMORY PURGE	要求分配器釋放內存
MEMORY STATS	顯示內存使用詳情
MEMORY USAGE key [SAMPLES count]	估計一個key的內存佔用
MGET key [key ...]	獲取所有給定鍵的值
MIGRATE host port key	destination-db timeout [COPY] [REPLACE] [AUTH password] [AUTH2 username password] [KEYS key [key ...]]
MONITOR	實時監聽服務器收到的所有請求
MOVE key db	移動一個鍵到另一個數據庫

MSET key value [key value ...]	將多個鍵設置為多個值
MSETNX key value [key value ...]	僅當不存在任何鍵時才將多個鍵設置為多個值
MULTI	標記一個事務塊的開始
OBJECT ENCODING key	檢查 Redis 對象的內部編碼
OBJECT FREQ key	獲取Redis對象的對數訪問頻率計數器
OBJECT IDLETIME key	獲取自上次訪問 Redis 對象以來的時間
OBJECT REFCOUNT key	獲取鍵值的引用次數
OBJECT HELP	顯示有關不同子命令的有用文本
PERSIST key	從鍵中刪除過期時間
PEXPIRE key milliseconds [NX	XX
PEXPIREAT key milliseconds-timestamp [NX	XX
PEXPIRETIME key	以毫秒為單位獲取鍵的到期 Unix 時間戳
PFADD key [element [element ...]]	將指定的元素添加到指定的 HyperLogLog.
PFCOUNT key [key ...]	返回 HyperLogLog 在 key(s) 處觀察到的集合的近似基數
PFMERGE destkey sourcekey [sourcekey ...]	將 N 個不同的 HyperLogLog 合併為一個
PING [message]	Ping 服務器
PSETEX key milliseconds value	以毫秒為單位設置鍵值和過期時間
PSUBSCRIBE pattern [pattern ...]	偵聽發佈到與給定模式匹配的頻道的消息
PUBSUB CHANNELS [pattern]	列出活動頻道
PUBSUB NUMPAT	獲取唯一模式訂閱的計數
PUBSUB NUMSUB [channel [channel ...]]	獲取頻道的訂閱者數量
PUBSUB HELP	顯示有用的文字，關於輸出不同的子命令
PTTL key	以毫秒為單位獲取鍵的生存時間
PUBLISH channel message	向頻道發佈消息
PUNSUBSCRIBE [pattern [pattern ...]]	停止收聽發佈到與給定模式匹配的頻道的消息

QUIT	關閉連接
RANDOMKEY	從鍵空間返回一個隨機鍵
READONLY	啟用對集群副本節點連接的讀取查詢
READWRITE	禁用對集群副本節點連接的讀取查詢
RENAME key newkey	重命名一個鍵
RENAMENX key newkey	重命名鍵，僅當新鍵不存在時
RESET	重置連接
RESTORE key ttl serialized-value [REPLACE] [ABSTTL] [IDLETIME seconds] [FREQ frequency]	使用提供的序列化值創建一個鍵，之前使用 DUMP 獲得
ROLE	返回實例在複製上下文中的角色
RPOP key [count]	刪除並獲取列表中的最後一個元素
RPOPLPUSH source destination	刪除列表中的最後一個元素，將其添加到另一個列表中並返回它
LMOVE source destination LEFT	RIGHT LEFT
RPUSH key element [element ...]	將一個或多個元素附加到列表中
RPUSHX key element [element ...]	僅當列表存在時才將元素附加到列表中
SADD key member [member ...]	將一個或多個成員添加到集合中
SAVE	將數據集同步保存到磁盤
SCARD key	獲取集合中的成員數量
SCRIPT DEBUG YES	SYNC
SCRIPT EXISTS sha1 [sha1 ...]	檢查腳本緩存中是否存在腳本
SCRIPT FLUSH [ASYNC	SYNC]
SCRIPT KILL	終止當前正在執行的腳本
SCRIPT LOAD script	將指定的 Lua 腳本加載到腳本緩存中
SDIFF key [key ...]	減去多組
SDIFFSTORE destination key [key ...]	減去多個集合並將結果集合存儲在一個鍵中

SELECT index	更改當前連接選擇的數據庫
SET key value [EX seconds	PX milliseconds
SETBIT key offset value	設置或清除存儲在 key
SETEX key seconds value	設置一個鍵的值和過期時間
SETNX key value	設置鍵的值，僅當鍵不存在時
SETRANGE key offset value	從指定的偏移量開始覆蓋鍵處的部分字符串
SHUTDOWN [NOSAVE	SAVE]
SINTER key [key ...]	返回由所有給定集合的交集產生的集合的成員
SINTERCARD numkeys key [key ...] [LIMIT limit]	將多個集合相交並返回結果的基數
SINTERSTORE destination key [key ...]	將多個集合相交並將結果集存儲在一個鍵中
SISMEMBER key member [member ...]	返回每個成員是否為存儲在key處的集合的成員
REPLICAOF host port	使服務器成為另一個實例的副本，或將其提升為主服務器
SLOWLOG GET [count]	獲取慢日誌的條目
SLOWLOG LEN	獲取慢日誌的長度
SLOWLOG RESET	清除慢日誌中的所有條目
SLOWLOG HELP	顯示有關不同子命令的有用文本
SMEMBERS key	獲取所有成員
SMOVE source destination member	將成員從一組移動到另一組
SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC	DESC] [ALPHA] [STORE destination]
SORT_RO key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC	DESC] [ALPHA]
SPOP key [count]	從集合中刪除並返回一個或多個隨機成員
SRANDMEMBER key [count]	從集合中獲取一個或多個隨機成員
SREM key member [member ...]	從集合中刪除一個或多個成員
STRLEN key	獲取存儲在鍵中的值的長度

SUBSCRIBE channel [channel ...]	收聽發佈到給定頻道的消息
SUNION key [key ...]	返回所有給定集合的並集所產生的集合的成員
SUNIONSTORE destination key [key ...]	返回所有給定集合的並集並存儲到指定集合中
SWAPDB index1 index2	交換兩個Redis數據庫
SYNC	內部命令，從主站發起複製流
PSYNC replicationid offset	內部命令，從主站發起複製流
TIME	返回當前服務器時間
TOUCH key [key ...]	更改鍵的最後訪問時間，返回指定的現有鍵的數量
TTL key	獲取鍵的剩餘生存時間（秒）
TYPE key	確定存儲在 key 的類型
UNSUBSCRIBE [channel [channel ...]]	停止收聽發佈到給定頻道的消息
UNLINK key [key ...]	在另一個線程中異步刪除一個鍵，否則它就像 DEL 一樣，但非阻塞
UNWATCH	取消監視所有鍵
WAIT numreplicas timeout	等待同步複製當前連接上下文中發送的所有寫命令
WATCH key [key ...]	觀察給定的鍵以確定 MULTI/EXEC 塊的執行
ZADD key [NX	XX] [GT
ZCARD key	獲取有序集合中的成員數量
ZCOUNT key min max	用給定值內的分數計算有序集中的成員
ZDIFF numkeys key [key ...] [WITHSCORES]	減去多個有序集
ZDIFFSTORE destination numkeys key [key ...]	減去多個有序集並將結果有序集存儲在一個新鍵中
ZINCRBY key increment member	在有序集中增加成員的分數
ZINTERCARD numkeys key [key ...] [LIMIT limit]	將多個有序集合相交並返回結果的基數
ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM	MIN
ZLEXCOUNT key min max	計算給定字典範圍內有序集中的成員數量

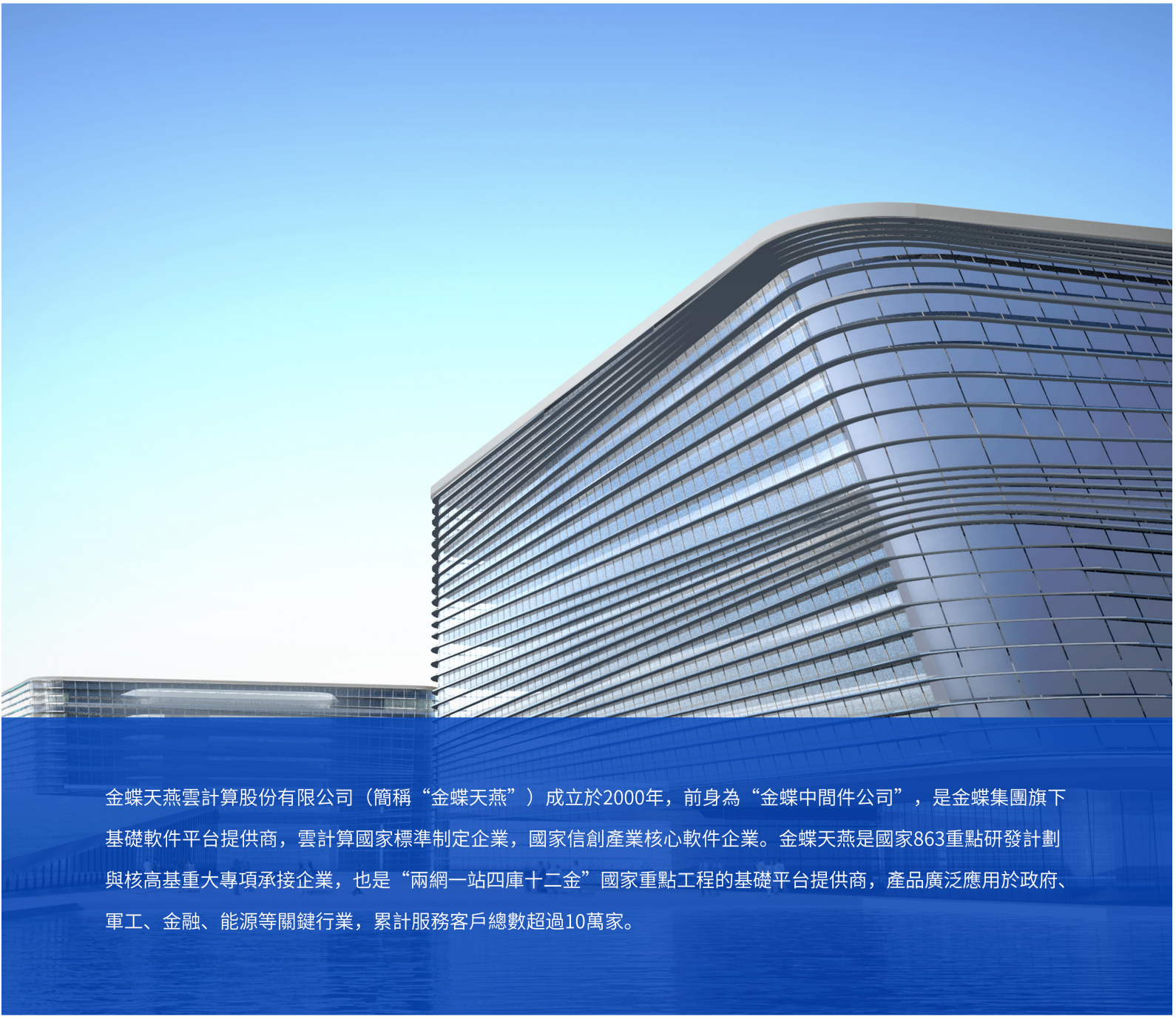
ZPOPMAX key [count]	刪除並返回有序集中得分最高的成員
ZPOPMIN key [count]	刪除並返回有序集中得分最低的成員
ZMPOP numkeys key [key ...] MIN	MAX [COUNT count]
ZRANDMEMBER key [count [WITHSCORES]]	從有序集合中獲取一個或多個隨機元素
ZRANGESTORE dst src min max [BYSCORE	BYLEX] [REV] [LIMIT offset count]
ZRANGE key min max [BYSCORE	BYLEX] [REV] [LIMIT offset count] [WITHSCORES]
ZRANGEBYLEX key min max [LIMIT offset count]	返回成員範圍rs 在一個有序的集合中，按字典序排列
ZREVRANGEBYLEX key max min [LIMIT offset count]	返回有序集中的成員範圍，按字典序範圍，從高到低的字符串排序
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	按分數返回有序集中的一系列成員
ZRANK key member	確定有序集合中成員的索引
ZREM key member [member ...]	從有序集中刪除一個或多個成員
ZREMRANGEBYLEX key min max	刪除給定字典範圍之間有序集中的所有成員
ZREMRANGEBYRANK key start stop	刪除給定索引內有序集中的所有成員
ZREMRANGEBYSCORE key min max	刪除給定分數內有序集中的所有成員
ZREVRANGE key start stop [WITHSCORES]	按索引返回有序集中的一系列成員，分數從高到低排序
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]	按分數返回有序集中的一系列成員，分數從高到低排序
ZREVRANK key member	確定一個有序集合中某個成員的索引，分數從高到低排序
ZSCORE key member	在有序集中獲取與給定成員關聯的分數
ZMSCORE key member [member ...]	獲取與有序集中的給定成員相關聯的分數
ZUNIONSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM	MIN
SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]	增量迭代鍵空間

SSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代 Set 元素
HSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代哈希字段和關聯值
ZSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代已排序的集合元素和相關分數
XINFO CONSUMERS key groupname	列出消費者組中的消費者
XINFO GROUPS key	列出流的消費者組
XINFO STREAM key [FULL [COUNT count]]	獲取有關流的信息
XINFO HELP	顯示有關不同子命令的有用文本
XADD key [NOMKSTREAM] [MAXLEN	MINID [=
XTRIM key MAXLEN	MINID [=
XDEL key ID [ID ...]	從流中刪除指定的條目，返回實際刪除的項目數，如果某些 ID 不存在，則可能與傳遞的 ID 數不同
XRANGE key start end [COUNT count]	返回流中的一系列元素，其 ID 與指定的 ID 間隔相匹配
XREVRANGE key end start [COUNT count]	與 XRANGE 相比，以相反的順序（從較大到較小的 ID）返回流中的一系列元素，其中 ID 與指定的 ID 間隔匹配
XLEN key	返回流中的條目數
XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	返回多個流中從未見過的元素，其 ID 大於調用者為每個流報告的 ID，可以屏蔽
XGROUP CREATE key groupname id	$ [MKSTREAM]
XGROUP CREATECONSUMER key groupname consumername	在消費者組中創建消費者
XGROUP DELCONSUMER key groupname consumername	從消費者組中刪除消費者
XGROUP DESTROY key groupname	銷毀一個消費組
XGROUP SETID key groupname id	$
XGROUP HELP	顯示有關不同子命令的有用文本
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key	使用消費者組從流中返回新條目，或訪問給定消費者的待處理條目的歷史記錄，可以屏蔽

[key ...] ID [ID ...]	
XACK key group ID [ID ...]	將待處理消息標記為正確處理，有效地將其從消費者組的待處理條目列表中刪除，該命令的返回值是成功確認的消息數，即成功在 PEL 中解析的 ID 數量
XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]	在流消費者組的上下文中，此命令更改掛起消息的所有權，以便新的所有者是作為命令參數指定的消費者
XAUTOCLAIM key group consumer min-idle-time start [COUNT count] [JUSTID]	在流消費者組的上下文中，此命令自動更改掛起消息的所有權，以便新的所有者是作為命令參數指定的消費者
XPENDING key group [[IDLE min-idle-time] start end count [consumer]]	從流消費者組待處理條目列表中返回信息和條目，即獲取但從未確認的消息
XIDMPRECORD key pid iid stream-id	持久化流IDMP狀態

全國統一服務熱線

4008-555-800



金蝶天燕雲計算股份有限公司（簡稱“金蝶天燕”）成立於2000年，前身為“金蝶中間件公司”，是金蝶集團旗下基礎軟件平台提供商，雲計算國家標準制定企業，國家信創產業核心軟件企業。金蝶天燕是國家863重點研發計劃與核高基重大專項承接企業，也是“兩網一站四庫十二金”國家重點工程的基礎平台提供商，產品廣泛應用於政府、軍工、金融、能源等關鍵行業，累計服務客戶總數超過10萬家。

Apusic 金蝶天燕

雲計算國家標準制定企業

金蝶集團旗下基礎軟件企業

資訊技術應用創新核心企業

官網：www.apusic.com