Apusic 金蝶天燕

生成时间：2026-07-13

Development Manual

金蝶Apusic分布式缓存 V2.0.4

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 Preface

This document is the development guide for Apusic In-Memory Data Cache (AMDC) V2.0.4, helping users quickly learn how to develop using Apusic In-Memory Data Cache.

1.1 Target Audience

This document is intended for development engineers, software architects, and R&D managers.

1.2 Related Documentation

For more information about AMDC V2.0.4, please refer to the following AMDC V2.0.4 product manual documentation set:

No.	Document	Description
1	Apusic In-Memory Data Cache V2.0.4 Quick Start Guide	A brief introduction on how to quickly get started with AMDC.
2	Apusic In-Memory Data Cache V2.0.4 Installation Guide	Detailed introduction on how to install AMDC on various operating systems, AMDC service start/stop operations, and product registration process.
3	Apusic In-Memory Data Cache V2.0.4 Core User Manual	Detailed introduction on how to use, configure, and manage AMDC features and supporting tools.
4	Apusic In-Memory Data Cache V2.0.4 Console User Manual	Detailed instructions on using and operating AMDC console features.
5	Apusic In-Memory Data Cache V2.0.4 Development Guide	Detailed instructions for developing AMDC client applications using various programming languages.
6	Apusic In-Memory Data Cache V2.0.4 Migration Guide	Detailed instructions for migrating AMDC from historical versions to V2.0.4, as well as migrating from Redis to AMDC.
7	Apusic In-Memory Data Cache V2.0.4 Operations Guide	Detailed instructions for AMDC monitoring, operations, security hardening, and other operational procedures.
8	Apusic In-Memory Data Cache V2.0.4 Performance Tuning Guide	Detailed instructions for AMDC performance tuning.

1.3 Technical Support

AMDC products provide comprehensive technical support services. You can obtain technical support through the following channels:

- Website: www.apusic.com
- Phone: 400-855-5800
- Email: support@apusic.com
- Kingdee Cloud Community: https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN

When requesting technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and version
4. Product version number
5. Detailed information including logs, screenshots, etc. of any exceptions or errors

2 Client Application Development

We will use `Java`, `Python`, and `Go`, three commonly used languages, to demonstrate AMDC's compatibility with various Redis clients.

2.1 Java Application Development

For Java development of Redis client applications, Jedis is recommended. Jedis is a lightweight and high-performance synchronous Java client officially recommended by Redis. It allows Java developers to conveniently and efficiently connect to, operate, and manage Redis databases in a way close to native commands.

Below are development examples for connecting to AMDC cache core services deployed in standalone, sentinel, and cluster modes using Jedis and performing data read/write operations.

- Standalone Node Connection and Usage

```
Jedis amdc = new Jedis("172.24.4.212", 6359);
System.out.println(amdc.ping());
System.out.println(amdc.set("key11", "value11"));
System.out.println(amdc.get("key11"));
```

- Sentinel Mode Connection and Usage

```
Set<String> sentinels = new HashSet<String>(Arrays.asList(
    "172.24.6.110:27000",
    "172.24.6.110:27001",
    "172.24.6.110:27002"
));
JedisSentinelPool pool = new JedisSentinelPool("mymaster",
sentinels, jedisPoolConfig);
Jedis jedis = pool.getResource();
System.out.println(jedis.set("key12", "value12"));
System.out.println(jedis.get("key12"));
```

- Cluster Mode Connection and Usage

```
Set<HostAndPort> clusterNodes = new HashSet<>();
clusterNodes.add(new HostAndPort("172.24.6.110", 7000));
clusterNodes.add(new HostAndPort("172.24.6.110", 7001));
clusterNodes.add(new HostAndPort("172.24.6.110", 7002));
clusterNodes.add(new HostAndPort("172.24.6.110", 7003));
clusterNodes.add(new HostAndPort("172.24.6.110", 7004));
clusterNodes.add(new HostAndPort("172.24.6.110", 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 Project Integration

AMDC is compatible with Redis data storage protocol, external clients can seamlessly switch to use AMDC product.

2.1.1.1 Create Project and Add Configuration

```
# application.yml
spring:
  redis:
    host: localhost       # Redis address
    port: 6379            # Port
    timeout: 2000ms       # Connection timeout
    database: 0           # Use database 0 (0-15)

    # Connection pool configuration (important!)
    lettuce:
      pool:
        max-active: 16    # Maximum connections
        max-idle: 8       # Maximum idle connections
```

```
        min-idle: 4       # Minimum idle connections
        max-wait: 1000ms # Maximum wait time to get connection
```

2.1.1.2 Configure RedisTemplate

```
@Configuration
public class RedisConfig {

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

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

        // Use String serialization for Key (important!)
        template.setKeySerializer(new StringRedisSerializer());
        template.setHashKeySerializer(new StringRedisSerializer());

        // Use JSON serialization for Value
        template.setValueSerializer(new
Jackson2JsonRedisSerializer<>(Object.class));
        template.setHashValueSerializer(new
Jackson2JsonRedisSerializer<>(Object.class));

        return template;
    }
}
```

2.1.1.3 Usage Example

Below is an example of cache participating in approval process status management.

```
@Service
public class ApprovalService {
```

```
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    /**
     * Store approval status
     */
    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()); // Approval data retained for
7 days

        // Use Hash storage
        redisTemplate.opsForHash().putAll(key, data);

        // Set expiration time
        redisTemplate.expire(key, 7, TimeUnit.DAYS);
    }

    /**
     * Get approval progress
     */
    public Map<Object, Object> getApprovalProgress(String
approvalId) {
        String key = "approval:status:" + approvalId;
        return redisTemplate.opsForHash().entries(key);
    }

    /**
```

```
     * Batch get pending items
     */
    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 Cache Usage Example

Redisson is a Redis Java client and real-time data platform that provides developers with the most convenient and easy-to-use way. Through Redisson objects, it establishes an elegant abstraction layer between Java code and Redis, allowing developers to focus on business logic and data models. At the same time, Redisson greatly extends Redis functionality, implementing many distributed features not available natively, including distributed collections, distributed locks, distributed objects, distributed services, etc.

This section demonstrates AMDC's development support and compatibility features for Redisson framework, providing relevant code examples for reference.

2.1.2.1 Quick Start

First we need to configure dependencies

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

```yaml
# application.yml - Standalone mode
spring:
  redis:
    redisson:
      config: |
        singleServerConfig:
          address: "redis://127.0.0.1:6379"
          password: null
          database: 0
          # Connection pool settings
          connectionPoolSize: 64
          connectionMinimumIdleSize: 24
          # Timeout settings (milliseconds)
          connectTimeout: 10000
          timeout: 3000
          retryAttempts: 3
        # Use JSON serialization
        codec: !<org.redisson.codec.JsonJacksonCodec> {}
```

Below are usage methods and partial parameter configurations for integrating different cache modes in code.

```java
@Configuration
public class RedissonConfig {

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

        // Standalone mode
        config.useSingleServer()
              .setAddress("redis://127.0.0.1:6379")
              .setPassword("yourPassword")
              .setDatabase(0)
              .setConnectionPoolSize(64)
              .setConnectionMinimumIdleSize(10);
```

```
        // Cluster mode (recommended for financial government
production environments)
        // config.useClusterServers()
        //       .addNodeAddress(
        //           "redis://node1:6379",
        //           "redis://node2:6379",
        //           "redis://node3:6379"
        //       )
        //       .setScanInterval(2000); // Cluster status scan
interval

        // Sentinel mode
        // config.useSentinelServers()
        //       .setMasterName("mymaster")
        //       .addSentinelAddress(
        //           "redis://sentinel1:26379",
        //           "redis://sentinel2:26379"
        //       );

        // Serialization configuration
        config.setCodec(new JsonJacksonCodec());

        // Thread configuration
        config.setThreads(16);  // Number of threads processing
Redis responses
        config.setNettyThreads(32); // Netty I/O thread count

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

Similarly, we use the approval workflow example:

```
@Service
public class ApprovalWorkflowService {
```

```
    @Autowired
    private RedissonClient redisson;

    /**
     * Distributed approval process
     */
    public void processApproval(String applicationId,
                                String approver,
                                boolean approved) {

        // 1. Get process lock
        RLock flowLock = redisson.getLock("approval:flow:" +
applicationId);

        if (flowLock.tryLock()) {
            try {
                // 2. Get current process status
                RMap<String, String> flowState =
                    redisson.getMap("approval:state:" +
applicationId);

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

                // 3. Check if approval is possible
                if (!"PENDING".equals(status)) {
                    throw new IllegalStateException("Application
already processed");
                }

                // 4. Update approval status
                flowState.put("approver", approver);
                flowState.put("approved", String.valueOf(approved));
                flowState.put("approvalTime",
LocalDateTime.now().toString());
```

```
                flowState.put("status", approved ? "APPROVED" :
"REJECTED");

                // 5. Publish approval completion event
                RTopic topic =
redisson.getTopic("approval:completed");
                topic.publishAsync(new ApprovalEvent(applicationId,
approved));

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

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

    /**
     * Multi-level countersign (requires all approvers to agree)
     */
    public boolean multiSignApproval(String applicationId,
List<String> approvers) {
        String latchKey = "approval:latch:" + applicationId;
        RCountDownLatch latch =
redisson.getCountDownLatch(latchKey);

        // Initialize counter
        latch.trySetCount(approvers.size());

        // Each approver processes independently
        for (String approver : approvers) {
```

```
            // Asynchronously submit to approver
            submitToApprover(applicationId, approver, latchKey);
        }

        try {
            // Wait for all approvers to complete (maximum 1 hour)
            return latch.await(1, TimeUnit.HOURS);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return false;
        }
    }
}
```

2.2 Python Application Development

- Standalone Node Connection and Usage

```
import redis
import time

r = redis.Redis(host='172.24.4.212', 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"))
```

- Sentinel Mode Connection and Usage

```
from redis.sentinel import Sentinel

SENTINELADDRS = [("172.21.33.69", "26359"), ("172.21.33.69",
```

```
"26360"), ("172.21.33.69", "26361")]
def sentinel():
    amdc = redis.Sentinel(sentinels=SENTINELADDRS)
    # Get master instance
    master = amdc.master_for("mymaster")
    # Get slave instance
    slave = amdc.slave_for("mymaster")
    # Write to master instance
    print(master.set("key2"))
    time.sleep(2)
    # Get from slave instance
    print(slave.get("key2"))
```

- Cluster Mode Connection and Usage

```
from redis.cluster import RedisCluster

CLUSTERADDRS = [{"host":"172.21.33.69", "port":"6359"},
{"host":"172.21.33.69", "port":"6360"}, {"host":"172.21.33.69",
"port":"6361"}]
def cluster():

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

2.3 Go Application Development

- Standalone Node Connection and Usage

```
package main

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

```
)

func main() {
    rdb := redis.NewClient(&redis.Options{
        Addr: "172.21.33.69:6359",
    })
    if err := rdb.Ping().Err(); err != nil {
        fmt.Println("Ping error:", err)
    } else {
        fmt.Println("Ping:", rdb.Ping().Val())
    }
}
```

- Sentinel Mode Connection and Usage

```
package main

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

func main() {
    rdb := redis.NewFailoverClient(&redis.FailoverOptions{
        MasterName: "mymaster",
        SentinelAddrs: []string{"172.21.33.69:26359",
"172.21.33.69:26360", "172.21.33.69:26361"},
    })
    defer rdb.Close()
    _, err := rdb.Set("hahakey", "hahavalue", 0).Result()
    if err != nil {
        fmt.Println("Set error:", err)
    } else {
        fmt.Println("Set hahakey: hahavalue")
```

```
    }
}
```

- Cluster Mode Connection and Usage

```
package main

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

func main() {
    addrs := []string{"172.21.33.69:6359", "172.21.33.69:6360",
"172.21.33.69:6361"}
    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 AMDC Operation Commands

Command	Description
ACL LOAD	Reload ACL from configured ACL file
ACL SAVE	Save current ACL rules in configured ACL file
ACL LIST	List current ACL rules in ACL configuration file format
ACL USERS	List all configured ACL rule usernames

ACL GETUSER username	Get rules for specific ACL user
ACL SETUSER username [rule [rule ...]]	Modify or create rules for specific ACL user
ACL DELUSER username [username ...]	Delete specified ACL users and related rules
ACL CAT [categoryname]	List ACL categories or commands within category
ACL GENPASS [bits]	Generate pseudo-random secure password for ACL user
ACL WHOAMI	Return the name of the user associated with current connection
ACL LOG [count or RESET]	List latest ACL security events
ACL HELP	Display helpful text about different subcommands
APPEND key value	Append value to key
ASKING	Sent by cluster client after -ask redirection
AUTH [username] password	Authenticate to server
BGREWRITEAOF	Asynchronously rewrite append-only file
BGSAVE [SCHEDULE]	Asynchronously save dataset to disk
BITCOUNT key [start end [BYTE|BIT]]	Count set bits in string
BITFIELD key [GET encoding offset] [SET encoding offset value] [INCRBY encoding offset increment] [OVERFLOW WRAP|SAT|FAIL]	Perform arbitrary bitfield integer operations on string
BITFIELD_RO key GET encoding offset	Perform arbitrary bitfield integer operations on string, read-only variant
BITOP operation destkey key [key ...]	Perform bitwise operations between strings
BITPOS key bit [start [end [BYTE|BIT]]]	Find first bit set or cleared in string
BLPOP key [key ...] timeout	Blockingly remove and return first element
BRPOP key [key ...] timeout	Blockingly remove and return last element
BRPOPLPUSH source target timeout	Remove last element from list and insert at head of another list; if list has no elements,

	block until timeout or element available
BLMOVE source destination FROM-TOP|FROM-BOTTOM TO-TOP| TO-BOTTOM timeout	Blockingly return and remove first or last element stored in source list (head or tail depends on wherefrom parameter), and push that element to first or last element of destination list (head or tail depends on whereto parameter)
LMPOP numkeys key [key ...] LEFT|RIGHT [COUNT count]	Pop one or more elements from first non-empty list key in provided key name list
BLMPOP timeout numkeys key [key ...] LEFT|RIGHT [COUNT count]	Blockingly pop one or more elements from first non-empty list key in provided key name list; or block connection until another client pushes to it or until timeout
BZPOPMIN key [key ...] timeout	Blockingly remove and return member with lowest score from one or more sorted sets, or block connection until another client pushes to it or until timeout
BZPOPMAX key [key ...] timeout	Blockingly remove and return member with highest score from one or more sorted sets, or block connection until another client pushes to it or until timeout
CLIENT CACHING YES|NO	Instruct server to track or not track keys in next request
CLIENT ID	Return client ID of current connection
CLIENT INFO	Return information about current client connection
CLIENT KILL [ip:port] [ID client-id] [TYPE normal|master|slave|pubsub] [USER username] [ADDR ip:port] [LADDR ip:port] [SKIPME yes/no]	Kill client connection
CLIENT LIST [TYPE normal|master|replica|pubsub] [ID client-id [client-id ...]]	Get list of client connections
CLIENT GETNAME	Get current connection name
CLIENT GETREDIR	Get tracking notification redirect client ID (if any)
CLIENT UNPAUSE	Resume processing of paused clients

CLIENT PAUSE timeout [WRITE|ALL]	Stop processing client commands for a period of time
CLIENT REPLY ON|OFF|SKIP	Instruct server whether to reply to commands
CLIENT SETNAME connection-name	Set current connection name
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix [PREFIX prefix ...]] [BCAST] [OPTIN] [OPTOUT] [NOLOOP]	Enable or disable server-assisted client cache support
CLIENT TRACKINGINFO	Return information about server-assisted client cache for current connection
CLIENT UNBLOCK client-id [TIMEOUT|ERROR]	Unblock client blocked by blocking command from different connection
COMMAND	Get array of Redis command details
COMMAND COUNT	Get total number of Redis commands
COMMAND GETKEYS	Extract keys given full Redis command
COMMAND INFO command-name [command-name ...]	Get array of specific Redis command details
CONFIG GET parameter [parameter ...]	Get values of configuration parameters
CONFIG REWRITE	Overwrite configuration file with in-memory configuration
CONFIG SET parameter value [parameter value ...]	Set configuration parameter to given value
CONFIG RESETSTAT	Reset statistics returned by INFO
COPY source destination [DB destination-db] [REPLACE]	Copy a key
DBSIZE	Return number of keys in selected database
DEBUG OBJECT key	Get debugging information about key
DEBUG SEGFAULT	Make server crash
DECR key	Decrement integer value of key by one
DECRBY key decrement	Decrement integer value of key by given number
DEL key [key ...]	Delete a key

DISCARD	Discard all commands issued after MULTI
DUMP key	Return serialized version of value stored at specified key
ECHO message	Echo given string
EVAL script numkeys [key [key ...]] [arg [arg ...]]	Execute Lua script server-side
EVALSHA sha1 numkeys [key [key ...]] [arg [arg ...]]	Execute Lua script server-side
EXEC	Execute MULTI
EXISTS key [key ...]	Determine if key exists
EXPIRE key seconds [NX|XX|GT|LT]	Set key's time to live in seconds
EXPIREAT key timestamp [NX|XX|GT|LT]	Set key's expiration time to UNIX timestamp
EXPIRETIME key	Get expiration Unix timestamp of key
FAILOVER [TO host port [FORCE]] [ABORT] [TIMEOUT milliseconds]	Start coordinated failover between this server and one of its replicas
FLUSHALL [ASYNC|SYNC]	Remove all keys from all databases
FLUSHDB [ASYNC|SYNC]	Remove all keys from current database
GEOADD key [NX|XX] [CH] longitude latitude member [longitude latitude member ...]	Add one or more geospatial items to geospatial index represented by sorted set
GEOHASH key member [member ...]	Return members of geospatial index as standard geohash strings
GEOPOS key member [member ...]	Return longitude and latitude of members of geospatial index
GEODIST key member1 member2 [m|km|ft|mi]	Return distance between two members of geospatial index
GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC|DESC] [STORE key] [STOREDIST key]	Query sorted set representing geospatial index for members matching given maximum distance from point
GEORADIUSBYMEMBER key member radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC|DESC] [STORE key] [STOREDIST key]	Query sorted set representing geospatial index for members matching given maximum distance from member

GEOSEARCH key [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [WITHCOORD] [WITHDIST] [WITHHASH]	Query sorted set representing geospatial index for members within box or circle area
GEOSEARCHSTORE destination source [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [STOREDIST]	Query sorted set representing geospatial index for members within box or circle area and store result in another key
GET key	Get value of a key
GETBIT key offset	Return bit value at offset in string value stored at key
GETDEL key	Get value of key and delete key
GETRANGE key start end	Get substring of string stored at key
GETSET key value	Set string value of key and return its old value
HDEL key field [field ...]	Delete one or more hash fields
HELLO [protover [AUTH username password] [SETNAME clientname]]	Handshake with Redis
HEXISTS key field	Determine if hash field exists
HGET key field	Get value of hash field
HGETALL key	Get all fields and values in hash
HINCRBY key field increment	Increment integer value of hash field by given number
HINCRBYFLOAT key field increment	Increment float value of hash field by given amount
HKEYS key	Get all fields in hash
HLEN key	Get number of fields in hash
HMGET key field [field ...]	Get values of all given hash fields
HMSET key field value [field value ...]	Set multiple hash fields to multiple values
HSET key field value [field value ...]	Set string value of hash field

HSETNX key field value	Set value of hash field only if field does not exist
HRANDFIELD key [count [WITHVALUES]]	Get one or more random fields from hash
HSTRLEN key field	Get length of value of hash field
HVALS key	Get all values in hash
INCR key	Increment integer value of key by one
INCRBY key increment	Increment integer value of key by given amount
INCRBYFLOAT key increment	Increment float value of key by given amount
INFO [section]	Get information and statistics about server
LOLWUT [VERSION version]	Display some computer art and Redis version
KEYS pattern	Find all keys matching given pattern
LASTSAVE	Get UNIX timestamp of last successful save to disk
LINDEX key index	Get element from list by index
LINSERT key BEFORE|AFTER pivot element	Insert element before or after another element in list
LLEN key	Get length of list
LPOP key [count]	Remove and get first element in list
LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]	Return index of matching element in list
LPUSH key element [element ...]	Add one or more elements to list
LPUSHX key element [element ...]	Add elements to list only if list exists
LRANGE key start stop	Get range of elements from list
LREM key count element	Remove elements from list
LSET key index element	Set value of element in list by index
LTRIM key start stop	Trim list to specified range
MEMORY DOCTOR	Output memory problem report

MEMORY HELP	Display helpful text about different subcommands
MEMORY MALLOC-STATS	Display allocator internal statistics
MEMORY PURGE	Request allocator to release memory
MEMORY STATS	Display memory usage details
MEMORY USAGE key [SAMPLES count]	Estimate memory usage of key
MGET key [key ...]	Get values of all given keys
MIGRATE host port key| destination-db timeout [COPY] [REPLACE] [AUTH password] [AUTH2 username password] [KEYS key [key ...]]	Atomically transfer key from Redis instance to another
MONITOR	Listen for all requests received by server in real time
MOVE key db	Move key to another database
MSET key value [key value ...]	Set multiple keys to multiple values
MSETNX key value [key value ...]	Set multiple keys to multiple values only if none of the keys exist
MULTI	Mark start of transaction block
OBJECT ENCODING key	Inspect internal encoding of Redis object
OBJECT FREQ key	Get logarithmic access frequency counter of Redis object
OBJECT IDLETIME key	Get time since Redis object was last accessed
OBJECT REFCOUNT key	Get reference count of value of key
OBJECT HELP	Display helpful text about different subcommands
PERSIST key	Remove expiration time from key
PEXPIRE key milliseconds [NX|XX|GT|LT]	Set key's time to live in milliseconds
PEXPIREAT key milliseconds-timestamp [NX|XX|GT|LT]	Set key's expiration time to UNIX timestamp specified in milliseconds

PEXPIRETIME key	Get expiration Unix timestamp of key in milliseconds
PFADD key [element [element ...]]	Add specified elements to specified HyperLogLog
PFCOUNT key [key ...]	Return approximate cardinality of set(s) observed by HyperLogLog at key(s)
PFMERGE destkey sourcekey [sourcekey ...]	Merge N different HyperLogLogs into one
PING [message]	Ping server
PSETEX key milliseconds value	Set key value and expiration time in milliseconds
PSUBSCRIBE pattern [pattern ...]	Listen for messages published to channels matching given pattern
PUBSUB CHANNELS [pattern]	List active channels
PUBSUB NUMPAT	Get count of unique pattern pattern subscriptions
PUBSUB NUMSUB [channel [channel ...]]	Get number of subscribers for channels
PUBSUB HELP	Display helpful text about different subcommands
PTTL key	Get time to live of key in milliseconds
PUBLISH channel message	Publish message to channel
PUNSUBSCRIBE [pattern [pattern ...]]	Stop listening for messages published to channels matching given pattern
QUIT	Close connection
RANDOMKEY	Return random key from keyspace
READONLY	Enable read queries for cluster replica node connection
READWRITE	Disable read queries for cluster replica node connection
RENAME key newkey	Rename key
RENAMENX key newkey	Rename key only if new key does not exist

RESET	Reset connection
RESTORE key ttl serialized-value [REPLACE] [ABSTTL] [IDLETIME seconds] [FREQ frequency]	Create key using provided serialized value, previously obtained using DUMP
ROLE	Return instance's role in replication context
RPOP key [count]	Remove and get last element in list
RPOPLPUSH source destination	Remove last element in list, add it to another list and return it
LMOVE source destination LEFT|RIGHT LEFT|RIGHT	Pop element from list, push to another list and return it
RPUSH key element [element ...]	Append one or more elements to list
RPUSHX key element [element ...]	Append elements to list only if list exists
SADD key member [member ...]	Add one or more members to set
SAVE	Synchronously save dataset to disk
SCARD key	Get number of members in set
SCRIPT DEBUG YES|SYNC|NO	Set debug mode for executed scripts
SCRIPT EXISTS sha1 [sha1 ...]	Check if script exists in script cache
SCRIPT FLUSH [ASYNC|SYNC]	Remove all scripts from script cache
SCRIPT KILL	Terminate currently executing script
SCRIPT LOAD script	Load specified Lua script into script cache
SDIFF key [key ...]	Subtract multiple sets
SDIFFSTORE destination key [key ...]	Subtract multiple sets and store result set in key
SELECT index	Change selected database for current connection
SET key value [EX seconds|PX milliseconds|EXAT timestamp|PXAT milliseconds-timestamp|KEEPTTL] [NX|XX] [GET]	Set string value of key
SETBIT key offset value	Set or clear bit at offset in string value stored at key

SETEX key seconds value	Set key value and expiration time
SETNX key value	Set key value only if key does not exist
SETRANGE key offset value	Overwrite part of string at key starting at specified offset
SHUTDOWN [NOSAVE|SAVE]	Synchronously save dataset to disk then shut down server
SINTER key [key ...]	Return members of set resulting from intersection of all given sets
SINTERCARD numkeys key [key ...] [LIMIT limit]	Intersect multiple sets and return cardinality of result
SINTERSTORE destination key [key ...]	Intersect multiple sets and store result set in key
SISMEMBER key member [member ...]	Return whether each member is member of set stored at key
REPLICAOF host port	Make server replica of another instance, or promote it as master
SLOWLOG GET [count]	Get entries from slow log
SLOWLOG LEN	Get length of slow log
SLOWLOG RESET	Clear all entries from slow log
SLOWLOG HELP	Display helpful text about different subcommands
SMEMBERS key	Get all members in set
SMOVE source destination member	Move member from one set to another
SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC|DESC] [ALPHA] [STORE destination]	Sort elements in list, set or sorted set
SORT_RO key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC|DESC] [ALPHA]	Sort elements in list, set or sorted set, read-only variant of SORT
SPOP key [count]	Remove and return one or more random members from set
SRANDMEMBER key [count]	Get one or more random members from set
SREM key member [member ...]	Remove one or more members from set

STRLEN key	Get length of value stored at key
SUBSCRIBE channel [channel ...]	Listen for messages published to given channels
SUNION key [key ...]	Return members of set resulting from union of all given sets
SUNIONSTORE destination key [key ...]	Return union of all given sets and store in specified set
SWAPDB index1 index2	Swap two Redis databases
SYNC	Internal command initiating replication stream from master
PSYNC replicationid offset	Internal command initiating replication stream from master
TIME	Return current server time
TOUCH key [key ...]	Change last access time of keys, return number of specified existing keys
TTL key	Get time to live of key in seconds
TYPE key	Determine type stored at key
UNSUBSCRIBE [channel [channel ...]]	Stop listening for messages published to given channels
UNLINK key [key ...]	Delete key asynchronously in another thread, otherwise like DEL but non-blocking
UNWATCH	Forget all watched keys
WAIT numreplicas timeout	Wait for synchronous replication of all write commands sent in current connection context
WATCH key [key ...]	Watch given keys to determine execution of MULTI/EXEC block
ZADD key [NX|XX] [GT|LT] [CH] [INCR] score member [score member ...]	Add one or more members to sorted set, or update score if already exists
ZCARD key	Get number of members in sorted set
ZCOUNT key min max	Count members in sorted set with scores within given values

ZDIFF numkeys key [key ...] [WITHSCORES]	Subtract multiple sorted sets
ZDIFFSTORE destination numkeys key [key ...]	Subtract multiple sorted sets and store result sorted set in new key
ZINCRBY key increment member	Increment member's score in sorted set
ZINTERCARD numkeys key [key ...] [LIMIT limit]	Intersect multiple sorted sets and return cardinality of result
ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]	Intersect multiple sorted sets and store result sorted set in new key
ZLEXCOUNT key min max	Count number of members in sorted set within given lexicographical range
ZPOPMAX key [count]	Remove and return member with highest score in sorted set
ZPOPMIN key [count]	Remove and return member with lowest score in sorted set
ZMPOP numkeys key [key ...] MIN|MAX [COUNT count]	Remove and return members with scores from sorted set
ZRANDMEMBER key [count [WITHSCORES]]	Get one or more random elements from sorted set
ZRANGESTORE dst src min max [BYSCORE|BYLEX] [REV] [LIMIT offset count]	Store range of members from sorted set into another key
ZRANGE key min max [BYSCORE|BYLEX] [REV] [LIMIT offset count] [WITHSCORES]	Return range of members in sorted set
ZRANGEBYLEX key min max [LIMIT offset count]	Return range of members in sorted set by lexicographical range
ZREVRANGEBYLEX key max min [LIMIT offset count]	Return range of members in sorted set by lexicographical range, ordered from high to low strings
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	Return range of members in sorted set by score
ZRANK key member	Determine index of member in sorted set
ZREM key member [member ...]	Remove one or more members from sorted set

ZREMRANGEBYLEX key min max	Remove all members in sorted set between given lexicographical range
ZREMRANGEBYRANK key start stop	Remove all members in sorted set within given index range
ZREMRANGEBYSCORE key min max	Remove all members in sorted set within given score range
ZREVRANGE key start stop [WITHSCORES]	Return range of members in sorted set by index, scores ordered from high to low
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]	Return range of members in sorted set by score, scores ordered from high to low
ZREVRANK key member	Determine index of member in sorted set, scores ordered from high to low
ZSCORE key member	Get score associated with given member in sorted set
ZMSCORE key member [member ...]	Get scores associated with given members in sorted set
ZUNIONSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]	Add multiple sorted sets and store result sorted set in new key
SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]	Incrementally iterate keyspace
SSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterate Set elements
HSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterate hash fields and associated values
ZSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterate sorted set elements and associated scores
XINFO CONSUMERS key groupname	List consumers in consumer group
XINFO GROUPS key	List consumer groups for stream
XINFO STREAM key [FULL [COUNT count]]	Get information about stream
XINFO HELP	Display helpful text about different subcommands

XADD key [NOMKSTREAM] [MAXLEN|MINID [=|~] threshold [LIMIT count]] *|ID field value [field value ...]	Append new entry to stream
XTRIM key MAXLEN|MINID [=|~] threshold [LIMIT count]	Trim stream to (approximately if '~' is passed) specific size
XDEL key ID [ID ...]	Remove specified entries from stream, return number of items actually deleted, may differ from number of IDs passed if some IDs do not exist
XRANGE key start end [COUNT count]	Return range of elements in stream with IDs matching specified ID interval
XREVRANGE key end start [COUNT count]	Return range of elements in stream with IDs matching specified ID interval in reverse order (from greater to smaller ID) compared to XRANGE
XLEN key	Return number of entries in stream
XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	Return elements never seen before in multiple streams with IDs greater than IDs reported by caller for each stream, can block
XGROUP CREATE key groupname id|$ [MKSTREAM]	Create a consumer group
XGROUP CREATECONSUMER key groupname consumername	Create consumer in consumer group
XGROUP DELCONSUMER key groupname consumername	Delete consumer from consumer group
XGROUP DESTROY key groupname	Destroy a consumer group
XGROUP SETID key groupname id|$	Set consumer group to arbitrary last delivered ID value
XGROUP HELP	Display helpful text about different subcommands
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key [key ...] ID [ID ...]	Return new entries from stream using consumer group, or access history of pending entries for given consumer, can block
XACK key group ID [ID ...]	Mark pending message as correctly processed, effectively removing it from consumer group's

	pending entries list, command return value is number of messages successfully acknowledged, i.e. IDs we were actually able to resolve in PEL
XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]	In context of stream consumer group, this command changes ownership of pending message so new owner is consumer specified as command argument
XAUTOCLAIM key group consumer min-idle-time start [COUNT count] [JUSTID]	In context of stream consumer group, this command automatically changes ownership of pending message so new owner is consumer specified as command argument
XPENDING key group [[IDLE min-idle-time] start end count [consumer]]	Return information and entries from stream consumer group pending entries list, i.e. messages that were fetched but never acknowledged

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com