Generated Date: 2026-10-23

Development Guide

Apusic Key-Value Database

Copyright © Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen 2026. All Rights Reserved.

Copyright Notice

All intellectual property rights including software copyrights involved in this document have been legally registered and duly owned by Kingdee Apusic Cloud Computing Co., Ltd. Such rights are protected by Copyright Law of the People's Republic of China, Regulations on Protection of Computer Software, Regulations on Intellectual Property Protection, relevant international copyright conventions, laws, regulations and other intellectual property legislations. Unauthorized use is strictly prohibited.

Disclaimer

All copyright information contained herein is legally owned by Kingdee Apusic Cloud Computing Co., Ltd. and protected by applicable laws. The Company shall bear no liability for any information unrelated to Kingdee Apusic contained in this document. You may view, copy and print this document only within the scope permitted by Copyright Law of the People's Republic of China. No entity or individual may use, modify or republish any part or content of this document without written authorization from Kingdee Apusic Cloud Computing Co., Ltd. Any violation shall constitute infringement, and the Company reserves the right to pursue legal liabilities accordingly.

Updates to this document shall not be notified separately. You may consult the Company regarding issues related to this document. All rights not expressly granted by the Company are reserved.

Trademark Notice

Apusic is a registered trademark filed with the National Intellectual Property Administration of PRC by Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen. Exclusive rights to the registered trademark are legally owned by Kingdee Apusic and protected by law. Without written consent of Kingdee Apusic, no entity or individual shall use, copy, modify, disseminate, transcribe, or bundle and sell any part of this trademark with other products in any manner or for any reason. Kingdee Apusic shall pursue legal liabilities against any infringement of its trademark rights. All other trademarks or registered trademarks mentioned in this document belong to their respective owners.

Contents

- 1 Preface
  - 1.1 Target Audience
  - 1.2 Related Documentation
  - 1.3 Technical Support
- 2 Client Application Development
  - 2.1 Java Application Development
    - 2.1.1 Spring Boot Project Integration
      - 2.1.1.1 Create a Project and Add Configuration
      - 2.1.1.2 Configure RedisTemplate
      - 2.1.1.3 Usage Example
    - 2.1.2 Redisson Cache Usage Example
      - 2.1.2.1 Quick Start
  - 2.2 Python Application Development
  - 2.3 Go Application Development
  - 2.4 KVDB Operation Commands

1 Preface

This document is the development guide for Apusic Key-Value Database (KVDB) V2.0, helping users quickly learn how to develop with Apusic Key-Value Database.

1.1 Target Audience

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

1.2 Related Documentation

For more information about Apusic KVDB V2.0, please refer to the following Apusic KVDB V2.0 product manual documentation set:

No.	Document	Description
1	Apusic Key-Value Database V2.0 Quick Start Guide	A brief introduction on how to quickly get started with KVDB.
2	Apusic Key-Value Database V2.0 Installation Guide	Detailed introduction on how to install KVDB on various operating systems, KVDB service start/stop operations, and the product registration process.
3	Apusic Key-Value Database V2.0 Key-Value Core User Manual	Detailed introduction on how to use, configure, and manage KVDB features and supporting tools.
4	Apusic Key-Value Database V2.0 Console User Manual	Detailed instructions on using and operating KVDB console features.
5	Apusic Key-Value Database V2.0 Development Guide	Detailed instructions for developing KVDB client applications using various programming languages.
6	Apusic Key-Value Database V2.0 Migration Guide	Detailed instructions for migrating KVDB from historical versions to V2.0, as well as migrating from Redis to KVDB.
7	Apusic Key-Value Database V2.0 Operations Guide	Detailed instructions for KVDB monitoring, operations, security hardening, and other operational procedures.
8	Apusic Key-Value Database V2.0 Performance Tuning Guide	Detailed instructions for KVDB performance tuning.

1.3 Technical Support

KVDB 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

This document uses three common languages — Java, Python, and Go — to demonstrate KVDB's compatibility with Redis clients in various languages.

2.1 Java Application Development

Jedis is recommended for developing Redis client applications in Java. Jedis is a lightweight, 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 that is close to native commands.

The following provides development examples of using Jedis to connect to key-value core services deployed in KVDB standalone, Sentinel, and cluster modes, and performing data read/write operations.

- Single-node connection and usage

```
Jedis kvdb = new Jedis("10.0.1.100 (example IP; replace with the actual IP)", 6359);
System.out.println(kvdb.ping());
System.out.println(kvdb.set("key11", "value11"));
System.out.println(kvdb.get("key11"));
```

- Sentinel mode connection and usage

```
Set<String> sentinels = new HashSet<String>(Arrays.asList(
    "10.0.1.110 (example IP; replace with the actual IP):27000",
    "10.0.1.110 (example IP; replace with the actual IP):27001",
    "10.0.1.110 (example IP; replace with the actual 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"));
```

- Cluster mode connection and usage

```
Set<HostAndPort> clusterNodes = new HashSet<>();
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual IP)", 7000));
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual IP)", 7001));
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual IP)", 7002));
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual IP)", 7003));
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual IP)", 7004));
clusterNodes.add(new HostAndPort("10.0.1.110 (example IP; replace
with the actual 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 Project Integration

KVDB is compatible with the Redis data storage protocol, so external clients can seamlessly switch to the KVDB product.

2.1.1.1 Create a Project and Add Configuration

```
# application.yml
spring:
  redis:
    host: localhost       # Redis address
    port: 6359           # Port
    timeout: 2000ms      # Connection timeout
```

```
    database: 0          # Use database 0 (0-15)

    # Connection pool configuration (important!)
    lettuce:
      pool:
        max-active: 16   # Maximum number of connections
        max-idle: 8      # Maximum number of idle connections
        min-idle: 4      # Minimum number of idle connections
        max-wait: 1000ms # Maximum wait time for obtaining a
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 keys (important!)
        template.setKeySerializer(new StringRedisSerializer());
        template.setHashKeySerializer(new StringRedisSerializer());

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

        return template;
```

```
    }
}
```

2.1.1.3 Usage Example

The following 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 is retained
for 7 days

        // Store using Hash
        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 Java client and real-time data platform for Redis, providing developers with the most convenient and easy-to-use approach. Through Redisson objects, it builds 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 the functionality of Redis, implementing many distributed features not available natively, including distributed collections, distributed locks, distributed objects, distributed services, and more.

This section demonstrates KVDB's development support and compatibility features for the Redisson framework, providing related code examples for reference.

2.1.2.1 Quick Start

First, configure the dependency

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

```
# application.yml - Standalone mode
spring:
  redis:
    redisson:
      config: |
        singleServerConfig:
          address: "redis://127.0.0.1:6359"
          password: "your_password"
          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> {}
```

The following shows how to use caches of different modes integrated in code, along with the configuration of some parameters.

```
@Configuration
public class RedissonConfig {

    @Bean(destroyMethod = "shutdown")
    public RedissonClient redissonClient() {
```

```
        Config config = new Config();

        // Standalone mode
        config.useSingleServer()
              .setAddress("redis://127.0.0.1:6359")
              .setPassword("your_password")
              .setDatabase(0)
              .setConnectionPoolSize(64)
              .setConnectionMinimumIdleSize(10);

        // Cluster mode (recommended for financial and government
production environments)
        // config.useClusterServers()
        //       .addNodeAddress(
        //           "redis://node1:6359",
        //           "redis://node2:6359",
        //           "redis://node3:6359"
        //       )
        //       .setScanInterval(2000); // Cluster status scan
interval

        // Sentinel mode
        // config.useSentinelServers()
        //       .setMasterName("mymaster")
        //       .addSentinelAddress(
        //           "redis://sentinel1:26359",
        //           "redis://sentinel2:26359"
        //       );

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

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

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

The following also uses the approval workflow example:

```
@Service
public class ApprovalWorkflowService {

    @Autowired
    private RedissonClient redisson;

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

        // 1. Acquire the workflow lock
        RLock flowLock = redisson.getLock("approval:flow:" +
applicationId);

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

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

                // 3. Check whether approval is allowed
                if (!"PENDING".equals(status)) {
```

```
                    throw new IllegalStateException("The application
has already been processed");
                }

                // 4. Update the 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 the approval completed event
                RTopic topic =
redisson.getTopic("approval:completed");
                topic.publishAsync(new ApprovalEvent(applicationId,
approved));

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

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

    /**
     * Multi-level countersignature (requires consent from all
approvers)
     */
    public boolean multiSignApproval(String applicationId,
```

```
List<String> approvers) {
        String latchKey = "approval:latch:" + applicationId;
        RCountDownLatch latch =
redisson.getCountDownLatch(latchKey);

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

        // Each approver processes independently
        for (String approver : approvers) {
            // Submit asynchronously to the approver
            submitToApprover(applicationId, approver, latchKey);
        }

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

2.2 Python Application Development

- Single-node connection and usage

```
import redis
import time

r = redis.Redis(host='10.0.1.100 (example IP; replace with the
actual 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"))
```

- Sentinel mode connection and usage

```
from redis.sentinel import Sentinel

SENTINELADDRS = [("10.0.1.120 (example IP; replace with the actual IP)", "26359"), ("10.0.1.120 (example IP; replace with the actual IP)", "26360"), ("10.0.1.120 (example IP; replace with the actual IP)", "26361")]
def sentinel():
    kvdb = redis.Sentinel(sentinels=SENTINELADDRS)
    # Get the master instance
    master = kvdb.master_for("mymaster")
    # Get the slave instance
    slave = kvdb.slave_for("mymaster")
    # Write to the master instance
    print(master.set("key2", "value2"))
    time.sleep(2)
    # Read from the slave instance
    print(slave.get("key2"))
```

- Cluster mode connection and usage

```
from redis.cluster import RedisCluster

CLUSTERADDRS = [{"host":"10.0.1.120 (example IP; replace with the actual IP)", "port":"6359"}, {"host":"10.0.1.120 (example IP; replace with the actual IP)", "port":"6360"}, {"host":"10.0.1.120 (example IP; replace with the actual 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 Application Development

- Single-node connection and usage

```
package main

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

func main() {
    rdb := redis.NewClient(&redis.Options{
        Addr: "10.0.1.120 (example IP; replace with the actual
IP):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{"10.0.1.120 (example IP; replace
with the actual IP):26359", "10.0.1.120 (example IP; replace with
the actual IP):26360", "10.0.1.120 (example IP; replace with the
actual 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")
    }
}
```

- Cluster mode connection and usage

```
package main

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

func main() {
    addrs := []string{"10.0.1.120 (example IP; replace with the
actual IP):6359", "10.0.1.120 (example IP; replace with the actual
IP):6360", "10.0.1.120 (example IP; replace with the actual
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 Operation Commands

Command	Description
ACL LOAD	Reloads the ACL rules from the configured ACL file
ACL SAVE	Saves the current ACL rules to the configured ACL file
ACL LIST	Lists the current ACL rules in ACL configuration file format
ACL USERS	Lists the usernames of all configured ACL rules
ACL GETUSER username	Gets the rules of a specific ACL user
ACL SETUSER username [rule [rule ...]]	Modifies or creates the rules of a specific ACL user
ACL DELUSER username [username ...]	Deletes the specified ACL users and their related rules
ACL CAT [categoryname]	Lists the ACL categories or the commands within a category
ACL GENPASS [bits]	Generates a pseudo-random secure password for an ACL user
ACL WHOAMI	Returns the name of the user associated with the current connection
ACL LOG [count or RESET]	Lists the latest ACL security events
ACL HELP	Displays helpful text about the different subcommands
APPEND key value	Appends a value to a key
ASKING	Sent by a cluster client after an ASK redirection
AUTH [username] password	Authenticates to the server
BGREWRITEAOF	Asynchronously rewrites the append-only file

BGSAVE [SCHEDULE]	Asynchronously saves the dataset to disk
BITCOUNT key [start end [BYTE|BIT]]	Counts the set bits in a string
BITFIELD key [GET encoding offset] [SET encoding offset value] [INCRBY encoding offset increment] [OVERFLOW WRAP|SAT|FAIL]	Performs arbitrary bitfield integer operations on strings
BITFIELD_RO key GET encoding offset	Performs arbitrary bitfield integer operations on strings; read-only variant with writes disabled
BITOP operation destkey key [key ...]	Performs bitwise operations between strings
BITPOS key bit [start [end [BYTE|BIT]]]	Finds the first bit set or cleared in a string
BLPOP key [key ...] timeout	Removes and returns the first element in a blocking manner
BRPOP key [key ...] timeout	Removes and returns the last element in a blocking manner
BRPOPLPUSH source target timeout	Pops the last element from a list and inserts it at the head of another list; if the list has no elements, it blocks until the wait times out or a poppable element is found.
BLMOVE source destination FROM-TOP|FROM-BOTTOM TO-TOP| TO-BOTTOM timeout	Atomically returns and removes the first or last element stored in the source list (head or tail depending on the wherefrom argument) in a blocking manner, and pushes that element to the first or last position of the target list (head or tail depending on the whereto argument)
LMPOP numkeys key [key ...] LEFT|RIGHT [COUNT count]	Pops one or more elements from the first non-empty list key among the provided key names
BLMPOP timeout numkeys key [key ...] LEFT|RIGHT [COUNT count]	Pops one or more elements from the first non-empty list key among the provided key names in a blocking manner; or blocks the connection until another client pushes to it or until the timeout expires
BZPOPMIN key [key ...] timeout	Removes and returns the member with the lowest score from one or more sorted sets in a blocking manner, or blocks the connection until another client pushes to it or until the timeout expires
BZPOPMAX key [key ...] timeout	Removes and returns the member with the highest score from one or more sorted sets in a blocking manner, or

	blocks the connection until another client pushes to it or until the timeout expires
CLIENT CACHING YES|NO	Instructs the server whether to track keys in the next request
CLIENT ID	Returns the client ID of the current connection
CLIENT INFO	Returns information about the 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]	Kills the connection of a client
CLIENT LIST [TYPE normal|master|replica|pubsub] [ID client-id [client-id ...]]	Gets the list of client connections
CLIENT GETNAME	Gets the current connection name
CLIENT GETREDIR	Gets the tracking notification redirection client ID (if any)
CLIENT UNPAUSE	Resumes processing of paused clients
CLIENT PAUSE timeout [WRITE|ALL]	Stops clients from processing commands for a period of time
CLIENT REPLY ON|OFF|SKIP	Instructs the server whether to reply to commands
CLIENT SETNAME connection-name	Sets the current connection name
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix [PREFIX prefix ...]] [BCAST] [OPTIN] [OPTOUT] [NOLOOP]	Enables or disables server-assisted client caching support
CLIENT TRACKINGINFO	Returns information about server-assisted client caching for the current connection
CLIENT UNBLOCK client-id [TIMEOUT|ERROR]	Unblocks a client blocked by a blocking command from a different connection
COMMAND	Gets an array of Redis command details
COMMAND COUNT	Gets the total number of Redis commands
COMMAND GETKEYS	Extracts keys given a full Redis command

COMMAND INFO command-name [command-name ...]	Gets an array of details for specific Redis commands
CONFIG GET parameter [parameter ...]	Gets the values of configuration parameters
CONFIG REWRITE	Rewrites the configuration file with the in-memory configuration
CONFIG SET parameter value [parameter value ...]	Sets a configuration parameter to the given value
CONFIG RESETSTAT	Resets the statistics returned by INFO
COPY source destination [DB destination-db] [REPLACE]	Copies a key
DBSIZE	Returns the number of keys in the selected database
DEBUG OBJECT key	Gets debugging information about a key
DEBUG SEGFAULT	Crashes the server
DECR key	Decrements the integer value of a key by one
DECRBY key decrement	Decrements the integer value of a key by the given number
DEL key [key ...]	Deletes a key
DISCARD	Discards all commands issued after MULTI
DUMP key	Returns a serialized version of the value stored at the specified key
DELEX key	Conditionally deletes a key based on the comparison result of the value (value equal/not equal, hash digest match/mismatch)
DIGEST key	Computes the digest of the value corresponding to a key (SM3/SHA256, etc.)
ECHO message	Echoes the given string
EVAL script numkeys [key [key ...]] [arg [arg ...]]	Executes a Lua script server-side
EVALSHA sha1 numkeys [key [key ...]] [arg [arg ...]]	Executes a Lua script server-side
EXEC	Executes MULTI

EXISTS key [key ...]	Determines whether a key exists
EXPIRE key seconds [NX	XX
EXPIREAT key timestamp [NX	XX
EXPIRETIME key	Gets the expiration Unix timestamp of a key
FAILOVER [TO host port [FORCE]] [ABORT] [TIMEOUT milliseconds]	Starts a coordinated failover between this server and one of its replicas
FLUSHALL [ASYNC	SYNC]
FLUSHDB [ASYNC	SYNC]
FUNCTION LOAD	Loads a server-side function (alternative to Lua scripts)
FUNCTION DELETE	Deletes the specified function
FUNCTION FLUSH	Clears all functions
FUNCTION LIST	Lists all functions
FUNCTION STATS	Views function execution statistics
GEOADD key [NX	XX] [CH] longitude latitude member [longitude latitude member ...]
GEOHASH key member [member ...]	Returns members of a geospatial index as standard geohash strings
GEOPOS key member [member ...]	Returns the longitude and latitude of members of a geospatial index
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	Gets the value of a key

GETBIT key offset	Returns the bit value at the offset in the string value stored at key
GETDEL key	Gets the value of a key and deletes the key
GETRANGE key start end	Gets a substring of the string stored at a key
GETSET key value	Sets the string value of a key and returns its old value
HDEL key field [field ...]	Deletes one or more hash fields
HELLO [protover [AUTH username password] [SETNAME clientname]]	Handshakes with Redis
HEXISTS key field	Determines whether a hash field exists
HGET key field	Gets the value of a hash field
HGETALL key	Gets all fields and values in a hash
HINCRBY key field increment	Increments the integer value of a hash field by the given number
HINCRBYFLOAT key field increment	Increments the float value of a hash field by the given amount
HKEYS key	Gets all fields in a hash
HLEN key	Gets the number of fields in a hash
HMGET key field [field ...]	Gets the values of all given hash fields
HMSET key field value [field value ...]	Sets multiple hash fields to multiple values
HSET key field value [field value ...]	Sets the string value of a hash field
HSETNX key field value	Sets the value of a hash field, only if the field does not exist
HRANDFIELD key [count [WITHVALUES]]	Gets one or more random fields from a hash
HSTRLEN key field	Gets the length of the value of a hash field
HVALS key	Gets all values in a hash
HOTKEYS [[value] [opt] ...]	Detects and lists current hot keys
INCR key	Increments the integer value of a key by one
INCRBY key increment	Increments the integer value of a key by the given amount

INCRBYFLOAT key increment	Increments the float value of a key by the given amount
INFO [section]	Gets information and statistics about the server
LOLWUT [VERSION version]	Displays some computer art and the Redis version
KEYS pattern	Finds all keys matching the given pattern
LASTSAVE	Gets the UNIX timestamp of the last successful save to disk
LINDEX key index	Gets an element from a list by its index
LINSERT key BEFORE	AFTER pivot element
LLEN key	Gets the length of a list
LPOP key [count]	Removes and gets the first elements in a list
LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]	Returns the index of matching elements in a list
LPUSH key element [element ...]	Adds one or more elements to a list
LPUSHX key element [element ...]	Adds elements to a list, only if the list exists
LRANGE key start stop	Gets a range of elements from a list
LREM key count element	Removes elements from a list
LSET key index element	Sets the value of an element in a list by its index
LTRIM key start stop	Trims a list to the specified range
MEMORY DOCTOR	Outputs a memory problem report
MEMORY HELP	Shows helpful text about the different subcommands
MEMORY MALLOC-STATS	Shows allocator internal statistics
MEMORY PURGE	Asks the allocator to release memory
MEMORY STATS	Shows memory usage details
MEMORY USAGE key [SAMPLES count]	Estimates the memory usage of a key
MGET key [key ...]	Gets the values of all given keys
MIGRATE host port key	destination-db timeout [COPY] [REPLACE] [AUTH password] [AUTH2 username password] [KEYS key [key ...]]

MONITOR	Listens in real time for all requests received by the server
MOVE key db	Moves a key to another database
MSET key value [key value ...]	Sets multiple keys to multiple values
MSETNX key value [key value ...]	Sets multiple keys to multiple values, only if none of the keys exist
MULTI	Marks the start of a transaction block
OBJECT ENCODING key	Inspects the internal encoding of a Redis object
OBJECT FREQ key	Gets the logarithmic access frequency counter of a Redis object
OBJECT IDLETIME key	Gets the time since a Redis object was last accessed
OBJECT REFCOUNT key	Gets the number of references of the value of a key
OBJECT HELP	Shows helpful text about the different subcommands
PERSIST key	Removes the expiration time from a key
PEXPIRE key milliseconds [NX	XX
PEXPIREAT key milliseconds-timestamp [NX	XX
PEXPIRETIME key	Gets the expiration Unix timestamp of a key in milliseconds
PFADD key [element [element ...]]	Adds the specified elements to the specified HyperLogLog.
PFCOUNT key [key ...]	Returns the approximated cardinality of the set(s) observed by the HyperLogLog at key(s)
PFMERGE destkey sourcekey [sourcekey ...]	Merges N different HyperLogLogs into a single one
PING [message]	Pings the server
PSETEX key milliseconds value	Sets the value and expiration of a key in milliseconds
PSUBSCRIBE pattern [pattern ...]	Listens for messages published to channels matching the given patterns
PUBSUB CHANNELS [pattern]	Lists the active channels
PUBSUB NUMPAT	Gets the count of unique pattern subscriptions
PUBSUB NUMSUB [channel [channel ...]]	Gets the number of subscribers of channels

PUBSUB HELP	Shows helpful text about the different subcommands
PTTL key	Gets the time to live of a key in milliseconds
PUBLISH channel message	Posts a message to a channel
PUNSUBSCRIBE [pattern [pattern ...]]	Stops listening for messages posted to channels matching the given patterns
QUIT	Closes the connection
RANDOMKEY	Returns a random key from the keyspace
READONLY	Enables read queries for a connection to a cluster replica node
READWRITE	Disables read queries for a connection to a cluster replica node
RENAME key newkey	Renames a key
RENAMENX key newkey	Renames a key, only if the new key does not exist
RESET	Resets the connection
RESTORE key ttl serialized-value [REPLACE] [ABSTTL] [IDLETIME seconds] [FREQ frequency]	Creates a key using the provided serialized value, previously obtained with DUMP
ROLE	Returns the role of the instance in the context of replication
RPOP key [count]	Removes and gets the last elements in a list
RPOPLPUSH source destination	Removes the last element in a list, prepends it to another list and returns it
LMOVE source destination LEFT	RIGHT LEFT
RPUSH key element [element ...]	Appends one or more elements to a list
RPUSHX key element [element ...]	Appends elements to a list, only if the list exists
SADD key member [member ...]	Adds one or more members to a set
SAVE	Synchronously saves the dataset to disk
SCARD key	Gets the number of members in a set
SCRIPT DEBUG YES	SYNC
SCRIPT EXISTS sha1 [sha1 ...]	Checks existence of scripts in the script cache

SCRIPT FLUSH [ASYNC	SYNC]
SCRIPT KILL	Kills the currently executing script
SCRIPT LOAD script	Loads the specified Lua script into the script cache
SDIFF key [key ...]	Subtracts multiple sets
SDIFFSTORE destination key [key ...]	Subtracts multiple sets and stores the resulting set in a key
SELECT index	Changes the selected database for the current connection
SET key value [EX seconds	PX milliseconds
SETBIT key offset value	Sets or clears the bit at offset in the string value stored at key
SETEX key seconds value	Sets the value and expiration of a key
SETNX key value	Sets the value of a key, only if the key does not exist
SETRANGE key offset value	Overwrites part of a string at key starting at the specified offset
SHUTDOWN [NOSAVE	SAVE]
SINTER key [key ...]	Returns the members of the set resulting from the intersection of all given sets
SINTERCARD numkeys key [key ...] [LIMIT limit]	Intersects multiple sets and returns the cardinality of the result
SINTERSTORE destination key [key ...]	Intersects multiple sets and stores the resulting set in a key
SISMEMBER key member [member ...]	Returns whether each member is a member of the set stored at key
REPLICAOF host port	Makes the server a replica of another instance, or promotes it as master
SLOWLOG GET [count]	Gets the entries of the slow log
SLOWLOG LEN	Gets the length of the slow log
SLOWLOG RESET	Clears all entries in the slow log
SLOWLOG HELP	Shows helpful text about the different subcommands
SMEMBERS key	Gets all the members in a set

SMOVE source destination member	Moves a member from one set to another
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]	Removes and returns one or more random members from a set
SRANDMEMBER key [count]	Gets one or more random members from a set
SREM key member [member ...]	Removes one or more members from a set
STRLEN key	Gets the length of the value stored in a key
SUBSCRIBE channel [channel ...]	Listens for messages published to the given channels
SUNION key [key ...]	Returns the members of the set resulting from the union of all given sets
SUNIONSTORE destination key [key ...]	Returns the union of all given sets and stores the result in the destination key
SWAPDB index1 index2	Swaps two Redis databases
SYNC	Internal command used for replication, initiated from the master
PSYNC replicationid offset	Internal command used for replication, initiated from the master
TIME	Returns the current server time
TOUCH key [key ...]	Changes the last access time of keys; returns the number of specified existing keys
TTL key	Gets the remaining time to live of a key (in seconds)
TYPE key	Determines the type stored at key
UNSUBSCRIBE [channel [channel ...]]	Stops listening for messages posted to the given channels
UNLINK key [key ...]	Deletes a key asynchronously in another thread; otherwise it is just like DEL, but non-blocking
UNWATCH	Forgets about all watched keys

WAIT numreplicas timeout	Waits for the synchronous replication of all the write commands sent in the context of the current connection
WATCH key [key ...]	Watches the given keys to determine execution of the MULTI/EXEC block
ZADD key [NX	XX] [GT
ZCARD key	Gets the number of members in a sorted set
ZCOUNT key min max	Counts the members in a sorted set with scores within the given values
ZDIFF numkeys key [key ...] [WITHSCORES]	Subtracts multiple sorted sets
ZDIFFSTORE destination numkeys key [key ...]	Subtracts multiple sorted sets and stores the resulting sorted set in a new key
ZINCRBY key increment member	Increments the score of a member in a sorted set
ZINTERCARD numkeys key [key ...] [LIMIT limit]	Intersects multiple sorted sets and returns the cardinality of the result
ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM	MIN
ZLEXCOUNT key min max	Counts the number of members in a sorted set between a given lexicographical range
ZPOPMAX key [count]	Removes and returns the members with the highest scores in a sorted set
ZPOPMIN key [count]	Removes and returns the members with the lowest scores in a sorted set
ZMPOP numkeys key [key ...] MIN	MAX [COUNT count]
ZRANDMEMBER key [count [WITHSCORES]]	Gets one or more random elements from a sorted set
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]	Returns a range of members in a sorted set, by lexicographical ordering
ZREVRANGEBYLEX key max min [LIMIT offset	Returns a range of members in a sorted set, by

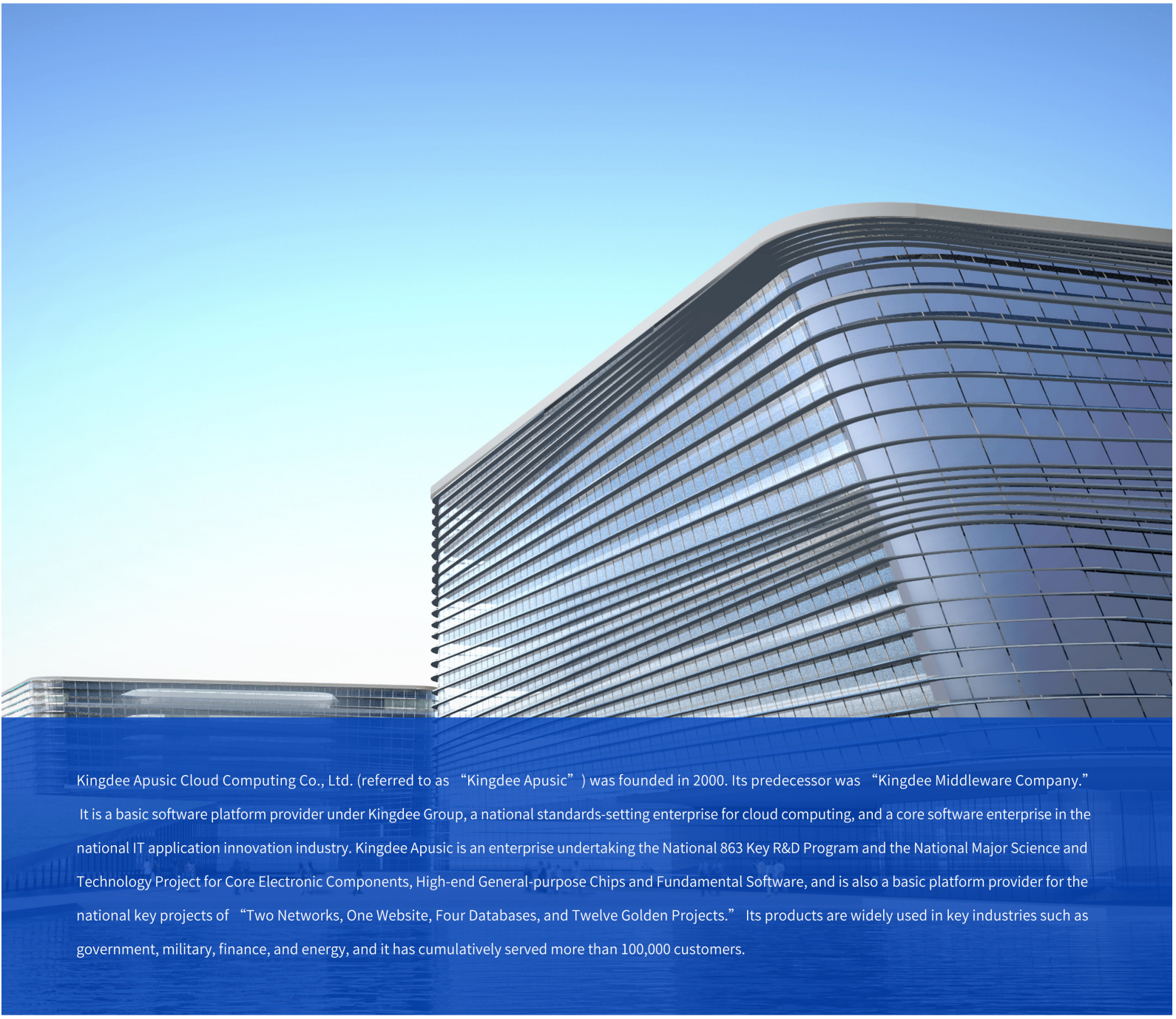
count]	lexicographical range, ordered from higher to lower strings
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	Returns a range of members in a sorted set, by score
ZRANK key member	Determines the index of a member in a sorted set
ZREM key member [member ...]	Removes one or more members from a sorted set
ZREMRANGEBYLEX key min max	Removes all members in a sorted set between the given lexicographical range
ZREMRANGEBYRANK key start stop	Removes all members in a sorted set within the given indexes
ZREMRANGEBYSCORE key min max	Removes all members in a sorted set within the given scores
ZREVRANGE key start stop [WITHSCORES]	Returns a range of members in a sorted set, by index, with scores ordered from high to low
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]	Returns a range of members in a sorted set, by score, with scores ordered from high to low
ZREVRANK key member	Determines the index of a member in a sorted set, with scores ordered from high to low
ZSCORE key member	Gets the score associated with the given member in a sorted set
ZMSCORE key member [member ...]	Gets the scores associated with the given members in a sorted set
ZUNIONSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM	MIN
SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]	Incrementally iterates the keyspace
SSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterates Set elements
HSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterates hash fields and associated values
ZSCAN key cursor [MATCH pattern] [COUNT count]	Incrementally iterates sorted set elements and associated scores

XINFO CONSUMERS key groupname	Lists the consumers in a consumer group
XINFO GROUPS key	Lists the consumer groups of a stream
XINFO STREAM key [FULL [COUNT count]]	Gets information about a stream
XINFO HELP	Shows helpful text about the different subcommands
XADD key [NOMKSTREAM] [MAXLEN	MINID [=
XTRIM key MAXLEN	MINID [=
XDEL key ID [ID ...]	Removes the specified entries from a stream; returns the number of items actually deleted, which may differ from the number of IDs passed if certain IDs do not exist
XRANGE key start end [COUNT count]	Returns a range of elements in a stream, with IDs matching the specified IDs interval
XREVRANGE key end start [COUNT count]	Compared to XRANGE, returns a range of elements in a stream in reverse order (from greater to smaller IDs), with IDs matching the specified IDs interval
XLEN key	Returns the number of entries in a stream
XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	Returns elements never seen by the caller from multiple streams, with IDs greater than the ones reported by the caller for each stream; can block
XGROUP CREATE key groupname id	$ [MKSTREAM]
XGROUP CREATECONSUMER key groupname consumername	Creates a consumer in a consumer group
XGROUP DELCONSUMER key groupname consumername	Deletes a consumer from a consumer group
XGROUP DESTROY key groupname	Destroys a consumer group
XGROUP SETID key groupname id	$
XGROUP HELP	Shows helpful text about the different subcommands
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key [key ...] ID [ID ...]	Returns new entries from a stream using a consumer group, or accesses the history of the pending entries for a given consumer; can block
XACK key group ID [ID ...]	Marks a pending message as correctly processed, effectively removing it from the pending entries list of the

	consumer group; the return value of the command is the number of messages successfully acknowledged, that is, the number of IDs resolved in the PEL
XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]	In the context of a stream consumer group, this command changes the ownership of a pending message, so that the new owner is the consumer specified as the command argument
XAUTOCLAIM key group consumer min-idle-time start [COUNT count] [JUSTID]	In the context of a stream consumer group, this command automatically changes the ownership of pending messages, so that the new owner is the consumer specified as the command argument
XPENDING key group [[IDLE min-idle-time] start end count [consumer]]	Returns information and entries from the pending entries list of a stream consumer group, that is, messages that were fetched but never acknowledged
XIDMPRECORD key pid iid stream-id	Persists the stream IDMP state

UNIFIED SERVICE HOTLINE

4008-555-800



Kingdee Apusic Cloud Computing Co., Ltd. (referred to as "Kingdee Apusic") was founded in 2000. Its predecessor was "Kingdee Middleware Company." It is a basic software platform provider under Kingdee Group, a national standards-setting enterprise for cloud computing, and a core software enterprise in the national IT application innovation industry. Kingdee Apusic is an enterprise undertaking the National 863 Key R&D Program and the National Major Science and Technology Project for Core Electronic Components, High-end General-purpose Chips and Fundamental Software, and is also a basic platform provider for the national key projects of "Two Networks, One Website, Four Databases, and Twelve Golden Projects." Its products are widely used in key industries such as government, military, finance, and energy, and it has cumulatively served more than 100,000 customers.

Apusic 金蝶天燕

National standard setter for cloud computing

Basic software enterprise under Kingdee Group

Core enterprise for IT application innovation

Website:www.apusic.com