Redis Tutorial

Introduction

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Redis Vs Memcached

Comparison Index	Memcached	Redis
What is	Memcached is in-memory key-value store, originally intended for caching purpose.	Redis is in-memory data structure store, used as database, cache and message broker.
Description	MemcacheD is easy, simply designed yet powerful. Its simple design promotes fast deployment, ease of exaggeration, and solves many problems related to large data caches. It has its inbuilt APIs which provide a very large hash table distributed across multiple machines & uses internal memory management to provide more efficiency. MemcacheD supports only String data type which are ideal for storing read-only data. Memcached is a volatile in-memory key-value origin. It is multi-threaded and used primarily for caching objects.	Redis is an open source in-memory data structure store which also can be used as a database as well as caching. It supports almost all types of data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes through radius queries. Redis also can be used for messaging system used as pub/sub.
Primary database model	Memcached follows Key-value store database model.	Redis also follows Key-value store database model.
Developed By	Memcached is developed by Danga Interactive.	Redis is developed by Salvatore Sanfilippo.

Initial release	Memcached was initially released in 2003.	Redis was initially released in 2009.
Current release	Memcached current version is 1.5.1, released in August 2017.	Redis current version is 4.0.2, released in September 2017.
License	Memcached is free and open source.	Redis is also free and open source.
Cloud-based	No	No
Implementation language	Memcached is implemented in C language.	Redis is also implemented in C language.
Server operating systems	FreeBSD Linux OS X Unix Windows	BSD Linux OS X Windows
Supported programming languages	.Net, C, C++, ColdFusion, Erlang, Java, Lisp, Lua, OCaml, Perl, PHP, Python, Ruby	C,C#,C++, Clojure, Crystal, D, Dart, Elixir, Erlang, Fancy, Go, Haskell, Haxe, Java, JavaScript (Node.js), Lisp, Lua, MatLab, Objective-C, OCaml info, Perl, PHP, Prolog, Pure Data, Python, R, Rebol, Ruby, Rust, Scala, Scheme, Smalltalk, Tcl
Server-side scripts	No	Lua
Triggers	No	No
Partitioning methods	None	Sharding
Replication methods	None	Master-slave replication
MapReduce	No	No
Foreign keys	No	No
Transaction concepts	No	Optimistic locking, atomic execution of commands blocks and scripts
Concurrency	Yes	Yes
Durability	No	Yes
User concepts	Yes	Simple password-based access control

Installation	Memcached is little bit complicated to install and run.	Installing Redis is so much easier. No dependencies required.
Memory Usage	MemcacheD is more memory efficient than Redis because it consumes comparatively less memory resources for metadata.	Redis is more memory efficient, only after you use Redis hashes.
Persistence	Memcached doesn't use persistent data. While using Memcached, data might be lost with a restart and rebuilding cache is a costly process.	Redis can handle persistent data. By default it syncs data to the disk at least every 2 seconds, offering optional & tuneable data persistence meant to bootstrap the cache after a planned shutdown or an unintentional failure. While we tend to regard the data in caches as volatile and transient, persisting data to disk can be quite valuable in caching scenarios.
Replication	Memcached does not support replication.	Redis supports master-slave replication.
Storage type	MemcacheD stores variables in its memory and retrieves any information directly from the server memory instead of hitting the database again.	Redis is like a database that resides in memory. It executes (reads and writes) a key/value pair from its database to return the result set. That's why it is used by develepors for real-time metrics and analytics.
Execution Speed and Performance	MemcacheD is very good to handle high traffic websites. It can read many information at a time and give you back at a great response time.	Redis can neither handle high traffic on read nor heavy writes.
Data Structure	MemcacheD uses only strings and integers in its data structure. So, everything you save can either be a string or an integer. It is complicated because with integers, the only data manipulation you can do is adding or subtracting them. If you need to save arrays or objects, you will have to serialize them first and then save	Redis has stronger data structures, which can handle not only strings integers but also binary-safe strings, lists of binary-safe strings, sets of binary-safe strings and sorted sets.

	them. To read them back, you will need to un-serialize.	
Key Length	Memcached's key length has a maximum of 250 bytes.	Redis' key-length has a maximum of 2GB.

Redis Vs MongoDB

Comparison Index	**Redis**	**MongoDB**
Introduction	Redis is in-memory data structure store, used as database, cache and message broker.	MongoDB is one of the most popular NoSQL database which follows the document stores structure.
Primary database model	Redis follows key-value store model.	MongoDB follows document store model.
Official Website	redis.io	www.mongodb.com
Technical Documentation	You can get technical documentation of Redis on redis.io/documentation	You can get technical documentation of MongoDB on docs.mongodb.com/manual
Developed By	Redis is developed by Salvatore Sanfilippo.	MongoDB is developed by MongoDB Inc.
Initial Release	Redis is initially released in 2009.	MongoDB is also initially released in 2009.
Licence	Redis is subscription based and open-source.	MongoDB is free to use and open-source.
Cloud based	No	No
Implementation Language	Redis is written and implemented in C language.	MongoDB is written and implemented in C++ language.
Server operating systems	BSD, Linux, OS X, Windows	Linux, OS X, Solaris, Windows
Data Scheme	schema-free	schema-free
Secondary Indexes	No	Yes

SQL	No	No
APIs and other access methods	Redis follows proprietary protocol.	MongoDB follows proprietary protocol using JSON.
Supported programming languages	C, C#, C++, Clojure, Crystal, D, Dart, Elixir, Erlang,Fancy, Go, Haskell, Haxe, Java, JavaScript (Node.js), Lisp, Lua, MatLab, Objective-C, OCaml, Perl, PHP, Prolog, Pure Data, Python, R, Rebol, Ruby, Rust, Scala, Scheme, Smalltalk, Tcl	Actionscript, C, C#, C++, Clojure, ColdFusion, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, MatLab Perl, PHP, PowerShell, Prolog, Python, R, Ruby, Scala, Smalltalk
Server-side scripts	Lua	JavaScript
Triggers	No	No
Partitioning methods	Redis uses Sharding for partition.	MongoDB also uses Sharding for partition.
Replication methods	Redis follows master-slave replication.	MongoDB also follows master-slave replication.
MapReduce	No	Yes
Consistency concepts	Eventual Consistency and Immediate Consistency	Eventual Consistency
Foreign keys	No	No
Transaction concepts	Optimistic locking, atomic execution of commands blocks and scripts.	No
Concurrency	Yes	Yes
MapReduce	No	Yes
Durability	Yes	Yes
In-memory capabilities	Yes	Yes
User concepts	Simple password-based access control.	Access rights for users and roles.

Special Characteristics	Redis is ranked as world?s fastest database. It reduces application complexity, simplifies development, accelerates time to market and provides unprecedented flexibility to developers with its visionary data structures and modules.	MongoDB is considered as the next-generation database. It successfully helped many businesses to transform their industries by providing big data. The world?s most sophisticated organizations, from cutting-edge startups to the largest companies, use MongoDB to create applications never before possible, at a very low cost.
Comparing Advantages	Redis is an in-memory database platform provides support of wide range of data structures such as strings, hashes, sets, lists, sorted sets, bitmaps, hyperloglogs, and geospatial indexes. Redis provides effortless scaling in a fully automated manner by overseeing all the operations of sharding, re-sharding, migration. It also includes persistence, instant automatic failure detection, backup and recovery, and in-memory replication across racks, zones, datacenters, regions, and cloud platforms.	MongoDB provides the best of traditional databases as well as flexibility, scale, and performance required by today?s applications. MongoDB is a database of giant ideas. MongoDB keeps the most valuable features of Relational database i.e. strong consistency, expressive query language and secondary indexes. It facilitates developers to build highly functional applications faster than NoSQL databases.
Key Customers	Key customers of Redis are: Verizon, Vodafone, Atlassian, Trip Advisor, Jet.com, Nokia, Samsung, HTC, Docker, Staples, Intuit, Groupon, Shutterfly, KPMG, TD Bank, UnitedHealthcare, RingCentral, The Motley Fool, Bleacher Report, HipChat, Salesforce, Hotel Tonight, Cirruspath, Itslearning.com, Xignite, Chargify, Rumble Entertainment, Scopely, Havas Digital, Revmob, MSN, Bleacher Report, Mobli, TMZ, Klarna, Shopify etc.	Key customers of MongoDB are: ADP, Adobe, AstraZeneca, BBVA, Bosch, Cisco, CERN, Department of Veteran Affairs, eBay, eHarmony, Electronic Arts, Expedia, Facebook?s Parse, Forbes, Foursquare, Genentech, MetLife, Pearson, Sage, Salesforce, The Weather Channel, Ticketmaster, Under Armour, Verizon Wireless etc.

Market Metrics	Redis Labs consists of more than 60000 customers globally and is consistently ranked as a leader in top analyst reports on NoSQL, in-memory and operational databases. Redis is rated as no. 1 cloud database, no.1 database in Docker, no.1 NoSQL datastore, most popular NoSQL database in container.	20 million downloads (growing at thousands downloads per day). More than 2,000 customers including over one third of the Fortune 100. Named a leader in the Forrester Wave?: Big Data NoSQL, Q3 2016. Highest placed non-relational database in DB Engines rankings

Redis Data Types

Data Types

Redis is not a plain key-value store, actually, it is a data structures server, supporting a different kind of values. In traditional key-value stores, you associated string keys to string values, in Redis the value is not limited to a simple string, but can also hold more complex data structures. The following is the list of all the data structures supported by Redis:

- **Binary-safe strings.**
- **Lists**
- **Sets**
- **Sorted sets**
- **Hashes**
- **Bit arrays (or simply bitmaps)**
- **HyperLogLogs:**

Redis keys:

Redis keys are binary safe (meaning they have a known length not determined by any special terminating characters), so you can use any binary sequence as a key, from a string like "foo" to the content of a JPEG file. The empty string is also a valid key. Here are some rules about keys:

- The maximum allowed key size is 512 MB.
- Very long keys are not a good idea.

- Very short keys are often not a good idea. While short keys will obviously consume a bit less memory, your job is to find the right balance.
- Try to stick with a schema. For instance "object-type:id" is a good idea, as in "user:1000". Dots or dashes are often used for multi-word fields, as in "comment:1234:reply.to" or "comment:1234:reply-to".

Redis Strings:

Strings are Redis' most basic data type. It is the only data type in Memcached, so it is also very natural for newcomers to use it in Redis. Since Redis keys are strings, when we use the string type as a value too, we are mapping a string to another string. The string data type is useful for a number of use cases, like caching HTML fragments or pages. Here are some common commands associated with strings:

- SET: sets a value to a key
- GET: gets a value from a key
- DEL: deletes a key and its value
- INCR: atomically increments a key
- INCRBY: increments a key by a designated values
- EXPIRE: the length of time that a key should exist (denoted in seconds)

Strings can be used to store objects, arranged by key.

```
127.0.0.1:6379>  SET newkey "the redis string"

OK

127.0.0.1:6379> GET newkey

"the redis string"
```

Redis Lists:

Lists in Redis are a collection of ordered values. This is in contrast to Sets which are unordered. Redis lists are implemented via Linked Lists. This means that even if you have millions of elements inside a list, the operation of adding a new element in the head or in the tail of the list is performed in constant time. Here are some common commands associated with lists:

- LPUSH: Add a value to the begriming of a list
- RPUSH: Add a value to the end of a list

- LPOP: Get and remove the first element in a list
- RPOP: Get and remove the last element in a list
- LREM: Remove elements from a list
- LRANGE: Get a range of elements from a list
- LTRIM: Modifies a list so leave only a specified range

Example:

```
redis 127.0.0.1:6379> lpush w3resourcelist redis

(integer) 1

redis 127.0.0.1:6379> lpush w3resourcelist mongodb

(integer) 2

redis 127.0.0.1:6379> lpush w3resourcelist rabitmq

(integer) 3

redis 127.0.0.1:6379> lrange w3resourcelist 0 10


1) "rabitmq"

2) "mongodb"

3) "redis"
```

Redis Sets:

Redis Sets are unordered collections of strings. If you want to combine strings, you can do that with REDIS sets. Here are some common commands associated with sets:

- SADD: Add one or members to a set
- SMEMBERS: Get all set members
- SINTER: Find the intersection of multiple sets
- SISMEMBER: check if a value is in a set
- SRANDMEMBER: Get a random set member

Sets can be helpful in various situations. In sets each member of a set is unique, adding members to a set does not require a "check then add" operation. Instead the set will check whether the item is a duplicate whenever an SADD command is performed.

Example:

```
redis 127.0.0.1:6379> sadd w3resourcelist redis

(integer) 1

redis 127.0.0.1:6379> sadd w3resourcelist mongodb

(integer) 1

redis 127.0.0.1:6379> sadd w3resourcelist rabitmq

(integer) 1

redis 127.0.0.1:6379> sadd w3resourcelist rabitmq

(integer) 0

redis 127.0.0.1:6379> smembers w3resourcelist


1) "rabitmq"

2) "mongodb"

3) "redis"
```

Redis Sorted sets:

Sorted sets are a data type which is similar to a mix between a Set and a Hash. Like sets, sorted sets are composed of unique, non-repeating string elements, so in some sense, a sorted set is a set as well.

However, while elements inside sets are not ordered, every element in a sorted set is associated with a floating point value, called the score (this is why the type is also similar to a hash, since every element is mapped to a value). Here are some common commands associated with sorted sets :

- ZADD: Adds members to a sorted set
- ZRANGE: Displays the members of a sorted set arranged by index (with the default low to high)
- ZREVRANGE: Displays the members of a sorted set arranged by index (from high to low)
- ZREM: Removes members from a sorted set

Example:

```
redis 127.0.0.1:6379> zadd w3resourcelist 0 redis

(integer) 1

redis 127.0.0.1:6379> zadd w3resourcelist 0 mongodb

(integer) 1

redis 127.0.0.1:6379> zadd w3resourcelist 0 rabitmq

(integer) 1

redis 127.0.0.1:6379> zadd w3resourcelist 0 rabitmq

(integer) 0

redis 127.0.0.1:6379> ZRANGEBYSCORE w3resourcelist 0 1000


1) "redis"

2) "mongodb"

3) "rabitmq"
```

Redis Hashes:

Hashes in Redis are useful to represent objects with many fields. They are set up to store a vast amount of fields in a small amount of space. Here are some common commands associated with hashes:

- HMSET: Sets up multiple hash values
- HSET: Sets the hash field with a string value
- HGET: Retrieves the value of a hash field
- HMGET: Retrieves all of the values for given hash fields
- HGETALL: Retrieves all of the values for in a hash

Example:

In the following example hash, data type is used to store user's object which contains basic information of a user.

```
redis 127.0.0.1:6379> HMSET user:1 username w3resource password 123456 points 200

OK

redis 127.0.0.1:6379> HGETALL user:1
```

1) "username"

2) "w3resource"

3) "password"

4) "123456"

5) "points"

6) "200"

Redis Bit arrays (or simply bitmaps):

It is possible, using special commands, to handle String values like an array of bits: you can set and clear individual bits, count all the bits set to 1, find the first set or unset bit, and so forth.

HyperLogLogs:

This is a probabilistic data structure which is used in order to estimate the cardinality of a set.

A complete Java example is given below.

```java
package com.ddlab.rnd;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
import redis.clients.jedis.Response;
import redis.clients.jedis.Transaction;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Test1 {

  public static void showRedisKeyExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    jedis.set("events/city/rome", "32,15,223,828");
    String cachedResponse = jedis.get("events/city/rome");
    System.out.println("cachedResponse = " + cachedResponse);
  }
  public static void showRedisListsExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    jedis.lpush("queue#tasks", "firstTask");
    jedis.lpush("queue#tasks", "secondTask");

    String task = jedis.rpop("queue#tasks");
    System.out.println("task = " + task);
    task = jedis.rpop("queue#tasks");
    System.out.println("task = " + task);
  }

  public static void showRedisSetsExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    jedis.sadd("nicknames", "nickname#1");
    jedis.sadd("nicknames", "nickname#2");
    jedis.sadd("nicknames", "nickname#1");

    Set<String> nicknames = jedis.smembers("nicknames");
    System.out.println("All Names : " +
jedis.smembers("nicknames"));
    System.out.println("nicknames = " + nicknames);
    boolean exists = jedis.sismember("nicknames", "nickname#1");
    System.out.println("exists = " + exists);
  }
```

```java
  public static void showRedisHashesExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    jedis.hset("user#1", "name", "Peter");
    jedis.hset("user#1", "job", "politician");
    String name = jedis.hget("user#1", "name");
    Map<String, String> fields = jedis.hgetAll("user#1");
    System.out.println("fields = " + fields);
    String job = fields.get("job");
    System.out.println("job = " + job);
  }

  public static void showRedisSortedSetExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    Map<String, Double> scores = new HashMap<>();

    scores.put("PlayerOne", 3000.0);
    scores.put("PlayerTwo", 1500.0);
    scores.put("PlayerThree", 8200.0);
    String key = "ranking";

    scores
        .entrySet()
        .forEach(
            playerScore -> {
              jedis.zadd(key, playerScore.getValue(),
playerScore.getKey());
            });

    String player = jedis.zrevrange("ranking", 0,
1).iterator().next();
    System.out.println("player = " + player);
    long rank = jedis.zrevrank("ranking", "PlayerOne");
    System.out.println("rank = " + rank);
  }

  public static void showRedisTransactionExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    String friendsPrefix = "friends#";
    String userOneId = "4352523";
    String userTwoId = "5552321";

    Transaction t = jedis.multi();
    t.sadd(friendsPrefix + userOneId, userTwoId);
    t.sadd(friendsPrefix + userTwoId, userOneId);
    t.exec();

    Set<String> nicknames = jedis.smembers(friendsPrefix +
userOneId);
    System.out.println("Txn : " + nicknames);
  }
  public static void showRedisPipelineExamples() {
    Jedis jedis = new Jedis("192.168.119.139", 6379);
    String userOneId = "4352523";
    String userTwoId = "4849888";

    Pipeline p = jedis.pipelined();
    p.sadd("searched#" + userOneId, "paris");
    p.zadd("ranking", 126, userOneId);
    p.zadd("ranking", 325, userTwoId);
    Response<Boolean> pipeExists = p.sismember("searched#" +
userOneId, "paris");
    System.out.println("pipeExists = " + pipeExists);
    Response<Set<String>> pipeRanking = p.zrange("ranking", 0, -1);
    p.sync();

    boolean exists = pipeExists.get();
    Set<String> ranking = pipeRanking.get();
    System.out.println("ranking = " + ranking);
  }

  public static void main(String[] args) {
    showRedisKeyExamples();
    showRedisListsExamples();
    showRedisSetsExamples();
    showRedisHashesExamples();
    showRedisSortedSetExamples();
    showRedisTransactionExamples();
    showRedisPipelineExamples();
  }
}
```