1 NoSQL

1.1 需求

高并发读写

海量数据的高效率存储与访问

高可扩展性和高可用性

1.2 产品分类

分类	Examples 举例	典型应用场景	数据模型	优点	缺点
键值(key-value)	Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB	内容缓存，主要用于处理大量数据的高访问负载，也用于一些日志系统等等	Key 指向 Value 的键值对，通常用 hash table 来实现	查找速度快	数据无结构化，通常只被当作字符串或者二进制数据
列存储数据库[	Cassandra, HBase, Riak	分布式的文件系统	以列簇式存储，将同一列数据存在一起	查找速度快，可扩展性强，更容易进行分布式扩展	功能相对局限
文档型数据库[	CouchDB, MongoDb	Web 应用（与 Key-Value 类似，Value 是结构化的，不同的是数据库能够了解 Value 的内容）	Key-Value 对应的键值对，Value 为结构化数据	数据结构要求不严格，表结构可变，不需要像关系型数据库一样需要预先定义表结构	查询性能不高，而且缺乏统一的查询语法。
图形(Graph)数据库	Neo4J, InfoGrid, Infinite Graph	社交网络，推荐系统等。专注于构建关系图谱	图结构	利用图结构相关算法。比如最短路径寻址，N 度关系查找等	很多时候需要对整个图做计算才能得出需要的信息，而且这种结构不太好做分布式的集群方案。

2 概述

2.1 Redis 简介

Remote Dictionary Server(Redis) 是使用 ANSI C 语言编写的 key-value 存储系统，支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。

它通常被称为数据结构服务器，因为值（value）可以是 字符串(String)， 哈希(Map)，列表(list)，集合(sets) 和 有序集合(sorted sets)等类型。

应用场景：缓存、任务队列、排行榜、网站访问统计、数据过期处理、分布式集群架构中的 session 分离

2.2 安装

2.2.1 下载

https://github.com/MSOpenTech/redis/releases

2.2.2 运行

1. 服务端

```
redis-server.exe redis.windows.conf
```

2. 客户端

```
redis-cli.exe -h 127.0.0.1 -p 6379
```

3 Jedis

Redis 的 Java 客户端开发包。

https://github.com/xetorthio/jedis

3.1 基本使用

```
Jedis jedis = new Jedis("127.0.0.1", 6379);
jedis.set("name", "rain");
String val = jedis.get("name");
System.out.println(val);
jedis.close();
```

3.2 连接池使用

```
JedisPoolConfig config = new JedisPoolConfig();
config.setMaxTotal(30);
config.setMaxIdle(10);

JedisPool jedisPool = new JedisPool(config, "127.0.0.1",
6379);
Jedis jedis = jedisPool.getResource();
jedis.set("date", "5/18");
String val = jedis.get("name");
System.out.println(val);
jedis.close();
jedisPool.close();
```

4 Key

操作	命令
查看	KEYS pattern 支持*、?
删除	DEL key1 [key2]
是否存在	EXISTS key
重命名	RENAME key newkey RENAMENX key newkey 仅当 newkey 不存在时，将 key 改名为 newkey
设置过期	EXPIRE key seconds
获取过期	TTL key 单位秒，-1 未设置过期 PTTL key 单位毫秒
值类型	TYPE key

5 Value 数据结构

5.1 String

二进制存储，最长 512M。

操作	命令
赋值	SET key value GETSET keye value
取值	GET key
删除	DEL key

增值	INCR key INCRBY key increment 不存在时，默认为 0
减值	DECR key DECRBY key increment 不存在时，默认为 0
拼接	APPEND key value 不存在时会创建

5.2 Hash

String 类型的 key 和 value 的映射表，适合用于存储对象。

每个 hash 可以存储 $2^{32} - 1$ 键值对（40 多亿）。

场景：存储一个用户信息对象数据。

操作	命令
赋值	HSET key field value HMSET key field1 value1 [field2 value2]
取值	HGET key field HMGET key field1 field2 HGETALL key
删除	HDEL key field1 [field2] DEL key
增值	HINCRBY key field increment
是否存在	HEXISTS key field
属性数量	HLEN key
属性列表	HKEYS key
值列表	HVALS key

5.3 List

简单的字符串列表，按照插入顺序排序。可添加元素到头部或尾部。

最多 $2^{32} - 1$ 个元素（40 多亿）。

底层数据结构是快速列表和压缩列表，快速列表是以压缩列表为节点的双向链表。

场景：轻松地实现最新消息排行。

操作	命令
赋值	LPUSH key value1 [value2] RPUSH key value1 [value2] LPUSHX key value(list 必须存在) RPUSHX key value(list 必须存在)
取值	LRANGE key start stop 负数表示从反方向开始

	LPOP key RPOP key
取值并赋值	RPOPLPUSH source destination
长度	LLEN
移除	LREM key count value (count：=0，全部；>0 正向；<0 逆向)
修改	LSET key index value
插入	LINSERT key BEFORE|AFTER pivot value

5.4 Set

String 类型的无序集合，无重复数据。

通过哈希表实现的，添加、删除、查找的复杂度都是 O(1)。

最多 2^{32} - 1 个元素 (40 多亿)。

场景：商品购买者交集

操作	命令
添加	SADD key member1 [member2]
删除	SREM key member1 [member2]
查看	SMEMBERS key SRANDMEMBER key [count] 随机查看，默认 1 个
是否存在	SISMEMBER key member
差集	SDIFF key1 [key2] key1 中元素除去 key2 中元素 SDIFFSTORE destination key1 [key2] 将差集存入 destination
交集	SINTER key1 [key2] SINTERSTORE destination key1 [key2] 将交集存入 destination
并集	SUNION key1 [key2] SUNIONSTORE destination key1 [key2] 将并集存入 destination
数量	SCARD key

5.5 SortedSet(ZSet)

每个元素关联一个 double 类型的分数，根据分数从小到大的排序，分数可以重复。

其余特性与 Set 一致。

场景：积分排行榜，构建索引。

操作	命令

添加	ZADD key score1 member1 [score2 member2]
获取	ZSCORE key member
查找	ZRANGE key start stop [WITHSCORES] 小到大 ZREVRANGE key start stop [WITHSCORES] 大到小 ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]
删除	ZREM key member [member ...] ZREMRANGEBYRANK key start stop 根据排名范围删除 ZREMRANGEBYSCORE key min max 根据分数删除
数量	ZCARD key 全部 ZCOUNT key min max 指定区间分数的成员数
修改	ZINCRBY key increment member 增加

6 特性

6.1 多数据库

单个实例最多支持 16 个数据库，下标从 0 开始，默认 0。

SELECT num，选择数据库。

MOVE key num，移动 key 到指定数据库。

6.2 事务

6.2.1 特征

事务中的命令串行化、顺序执行，执行期间不为其他客户端提供任何服务，保证所有命令被原子化执行。

事务中某命令执行失败，后续命令会继续执行。

6.2.2 操作

操作	命令
开始事务	MULTI
执行事务	EXEC
取消事务	DISCARD

7 持久化

7.1 方式

RDB、AOF、无持久化、RDB 与 AOF 结合

7.2 RDB

7.2.1 概述

默认方案。在指定的时间间隔内，执行指定次数的写操作，则会将内存中的数据写入到磁盘中。即在指定目录下生成一个 dump.rdb 文件。Redis 重启会通过加载 dump.rdb 文件恢复数据。

7.2.2 优劣

优点：适合大规模的数据恢复；如果业务对数据完整性和一致性要求不高，RDB 是很好的选择。

缺点：数据的完整性和一致性不高，因为 RDB 可能在最后一次备份时宕机了；备份时占用内存，因为 Redis 在备份时会独立创建一个子进程，将数据写入到一个临时文件（此时内存中的数据是原来的两倍），最后再将临时文件替换之前的备份文件。

7.2.3 配置

redis.conf 文件中 SNAPSHOTTING 部分

1. 规则配置

```
save <seconds> <changes>
# save ""
save 900 1
save 300 10
save 60 10000
```

2. 文件配置

```
dbfilename dump.rdb
```

3. 目录配置

```
dir ./
```

4. 数据压缩

```
rdbcompression yes
```

7.3 AOF

7.3.1 概述

默认不开启。它的出现是为了弥补 RDB 的不足（数据的不一致性），所以它采用日志的形式来记录每个写操作，并追加到文件中。Redis 重启会根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。

7.3.2 优劣

优点：数据的完整性和一致性更高

缺点：因为 AOF 记录的内容多，文件会越来越大，数据恢复也会越来越慢。

7.3.3 配置

1. 开启

```
appendonly yes
```

2. 文件配置

```
appendfilename "appendonly.aof"
```

3. 保存规则配置

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

4. 重写规则配置

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 3gb
```

8 参考资料

1. 《Redis 参考资料》 redis 官网 https://redis.io/documentation
2. 《Redis 命令》 redis 官网 https://redis.io/commands
3. 《Redis 教程》 菜鸟教材 http://www.runoob.com/redis/redis-tutorial.html
4. 《Redis 入门教程》 慕课网 https://www.imooc.com/view/839
5. 《Redis 持久化之 RDB 和 AOF》 博客园 https://www.cnblogs.com/itdragon/p/7906481.html