NoSQL之

Redis介绍及实践分享

Derbysoft 内部技术分享交流

目录

1.Redis是什么

1. Redis是REmote DIctionary Server的缩写，是一个key-value存储系统.
2. Redis提供了一些丰富的数据结构，包括Strings，Lists，Hashes，Sets和Ordered Sets以及Hashes. 包括对这些数据结构的操作支持.
3. Redis可以替代Memcached,并且解决了断电后数据完全丢失的问题.
4. Redis官方网站: http://redis.io
   Redis作者Blog: http://antirez.com

2.Redis安装

Download, extract and compile Redis with:

$ wget http://redis.googlecode.com/files/redis-2.4.5.tar.gz

$ tar xzf redis-2.4.5.tar.gz

$ cd redis-2.4.5

$ make

The binaries that are now compiled are available in the src directory. Run Redis with:

$ src/redis-server

You can interact with Redis using the built-in client:

$ src/redis-cli

redis> set foo bar

OK

redis> get foo

"bar"

3.Redis优点

1.性能极高,Redis能支持10万每秒的读写频率.

2.丰富的数据类型及对应的操作.

3.Redis的所有操作都是原子性的,同时Redis还支持对几个操作全并后的原子性执行, 也即支持事务.

4.丰富的特性,Redis还支持publish/subscribe, key过期等特性.

4.Redis性能

以下摘自官方测试描述:

在50个并发的情况下请求10W次,写的速度是11W次/s,读的速度是8.1w次/s.

测试环境:

1.50个并发,请求10W次.

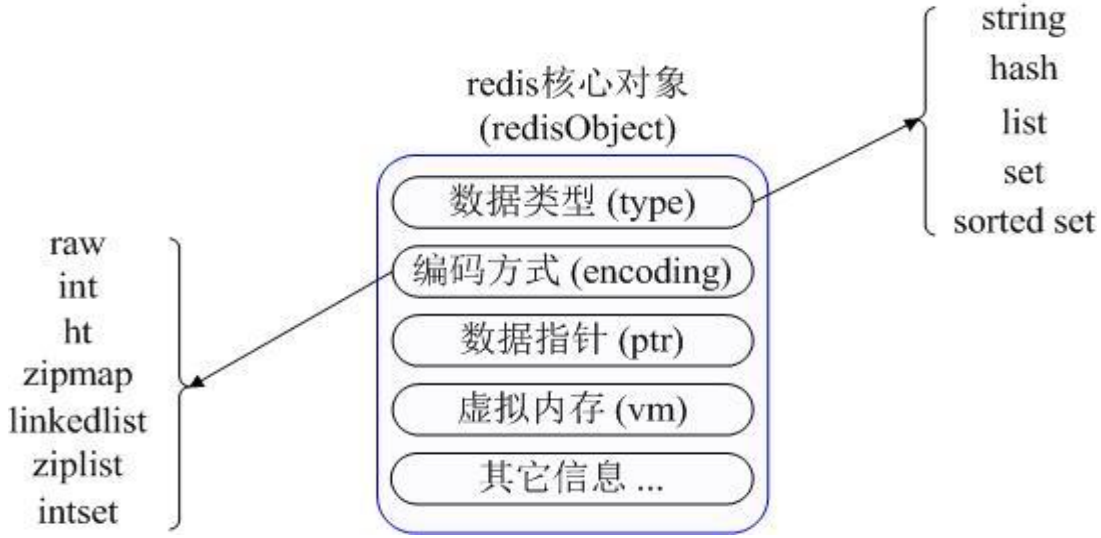
2.读和写大小为256bytes的字符串.

3.Linux2.6 Xeon X3320 2.5GHz的服务器上.

4.通过本机的loopback interface接口上执行.

5.Redis数据类型

redis常用五种数据类型:string,hash,list,set,sorted set.



Redis内部使用一个redisObject 对象来表示所有的key和value, redisObject最主要的信息如上图所示: type代表一个value对象具体是何种数据类型, encoding是不同数据类型在redis内部的存储方式, 比如: type=string代表value存储的是一个普通字符串, 那么对应的encoding可以是raw或者是int, 如果是int 则代表实际redis内部是按数值型类存储和表示这个字符串的.

Commands 集合: http://redis.io/commands

5.Redis数据类型

5.1 String

常用命令:

set,get,decr,incr,mget 等.

应用场景:

String是最常用的一种数据类型,普通的key/value存储.

实现方式:

String在redis内部存储默认就是一个字符串,被redisObject所引用,当遇到incr,decr等操作时会转成数值型进行计算,此时redisObject的encoding字段为int.

5.Redis数据类型

5.2 Hash

常用命令:

hget,hset,hgetall 等.

应用场景:

比如,我们存储供应商酒店价格的时候可以采取此结构,用酒店编码作为Key,RatePlan+RoomType作为Filed,价格信息作为Value.

实现方式:

Hash对应Value内部实际就是一个HashMap,实际这里会有2种不同实现,这个Hash的成员比较少时Redis为了节省内存会采用类似一维数组的方式来紧凑存储,而不会采用真正的HashMap结构,对应的value redisObject的encoding为zipmap,当成员数量增大时会自动转成真正的HashMap,此时encoding为ht.

5.Redis数据类型

5.3 List

常用命令:

lpush,rpush,lpop,rpop,lrang等.

应用场景:

Redis list应用场景非常多,也是Redis最重要的数据结构之一,比如twitter的关注列表,粉丝列表等都可以用Redis的list结构来实现.

实现方式:

Redis list的实现为一个双向链表,即可以支持反向查找和遍历,更方便操作,不过带来了部分额外的内存开销,Redis内部的很多实现,包括发送缓冲队列等也都是用的这个数据结构.

5.Redis数据类型

5.4 Set

常用命令:

sadd,spop,smembers,sunion 等.

应用场景:

Set对外提供的功能与list类似,当你需要存储一个列表数据,又不希望出现重复数据时,set 是一个很好的选择,并且set提供了判断某个成员是否在一个set集合内的接口,这个也是list所不能提供的.

实现方式:

Set 的内部实现是一个value永远为null的HashMap,实际就是通过计算hash的方式来快速排除重复的,这也是set能提供判断一个成员是否在集合内的原因.

5.Redis数据类型

5.5 Sorted set

常用命令:

zadd,zrange,zrem,zcard等.

使用场景:

Sorted set的使用场景与set类似,区别是set不是自动有序的,而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序,并且是插入有序的,即自动排序.当你需要一个有序的并且不重复的集合列表,那么可以选择sorted set数据结构.

实现方式:

Sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序,HashMap里放的是成员到score的映射,而跳跃表里存放的是所有的成员,排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率.

5.Redis数据类型

5.6 内存优化

Redis为不同数据类型分别提供了一组参数来控制内存使用,我们在前面提到过Redis Hash的value内部是一个HashMap,如果该Map的成员数比较少,则会采用一维数组的方式来紧凑存储该Map,即省去了大量指针的内存开销,这个参数在redis.conf配置文件中下面2项:

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

含义是当value这个Map内部不超过多少个成员时会采用线性紧凑格式存储,默认是64,即value内部有64个以下的成员就是使用线性紧凑存储,超过该值自动转成真正的HashMap.

hash-max-zipmap-value 含义是当value这个Map内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间.

以上2个条件任意一个条件超 过设置值都会转换成真正的HashMap,也就不会再节省内存了,那么这个值是不是设置的越大越好呢,答案当然是否定的,HashMap的优势就是查找和操作的时间复杂度都是O(1)的,而放弃Hash采用一维存储则是O(n)的时间复杂度,如果成员数量很少,则影响不大,否则会严重影响性能,所以要权衡好这个值的设置,总体上还是时间成本和空间成本上的权衡.

6.Redis发布/订阅

Redis的发布/订阅(Publish/Subscribe)功能类似于传统的消息路由功能,发布者发布消息,订阅者接收消息,沟通发布者和订阅者之间的桥梁是订阅的Channel或者Pattern.

订阅者和发布者之间的关系是松耦合的,发布者不指定哪个订阅者才能接收消息,订阅者不只接收特定发布者的消息.

7.Redis数据过期设置

Redis可以按key设置过期时间，过期后将被自动删除，这个特性让Redis很适合用来存储酒店动态流量和价格信息.

用TTL命令可以获取某个key值的过期时间，-1表示不会过期.

8.Redis事务支持

Redis目前对事务支持还比较简单,也即支持一些简单的组合型的命令,只能保证一个client发起的事务中的命令可以连续的执行,而中间不会插入其他client的命令. 由于Redis是单线程来处理所有client的请求的所以做到这点是很容易的.

事务的执行过程中,如果redis意外的挂了,这时候事务可能只被执行了一半,可以用redis-check-aof 工具进行修复.

9.Redis数据存储

9.1 数据快照

数据快照的原理是将整个Redis内存中存的所有数据遍历一遍存到一个扩展名为rdb的数据文件中. 通过SAVE命令可以调用这个过程.

9.Redis数据存储

9.2 数据快照配置

save 900 1

save 300 10

save 60 10000

以上在redis.conf中的配置指出在多长时间内, 有多少次更新操作, 就将数据同步到数据文件, 这个可以多个条件配合. 上面的含义是900秒后有一个key发生改变就执行save, 300秒后有10个key发生改变执行save, 60秒有10000个key发生改变执行save

10.Redis AOF

10.1 数据快照的缺点是持久化之后如果出现crash则会丢失一段数据，因此作者增加了另外一种追加式的操作日志记录，叫append only file，其日志文件以aof结尾，我们一般称为aof文件。要开启aof日志的记录，需要在配置文件中进行如下设置：

appendonly yes

10.Redis AOF

10.2 appendonly配置如果不开启,可能会在断电时导致一段时间内的数据丢失. 因为redis本身同步数据文件是按save条件来同步的, 所以有的数据会在一段时间内只存在于内存中.

appendfsync no/always/everysec
1. no:表示等操作系统进行数据缓存同步到磁盘.
2. always:表示每次更新操作后手动调用fsync() 将数据写到磁盘.
3. everysec:表示每秒同步一次. 一般用everysec.

10.Redis AOF BGREWRITEAOF

10.3 AOF文件只增不减会导致文件越来越大，重写过程如下

1. Redis通过fork产生子进程.
2. 子进程将当前所有数据写入一个临时文件.
3. 父子进程是并行执行的，在子进程遍历并写临时文件的时候，父进程在照常接收请求，处理请求，写AOF，不过这时他是把新来的AOF写在一个缓冲区中.
4. 子进程写完临时文件后就会退出. 这时父进程会接收到子进程退出的消息，他会把自己现在收集在缓冲区中的所有AOF追加在临时文件中.
5. 最后把临时文件rename一下，改名为appendonly.aof， 这时原来的aof文件被覆盖. 整个过程完成.

11.Redis数据恢复

当Redis服务器挂掉时，重启时将按以下优先级恢复数据到内存种：

1. 如果只配置了AOF,重启时加载AOF文件恢复数据.

2.如果同时配置了RBD和AOF，启动时只加载AOF文件恢复数据.

3.如果只配置了RDB,启动时将加载dump文件恢复数据.

12.Redis主从复制

12.1

Master/Slave配置:

Master IP:175.41.209.118

Master Redis Server Port:6379

Slave配置很简单,只需要在slave服务器的redis.conf加入:

slaveof 175.41.209.118 6379

启动master和slave,然后写入数据到master,读取slave,可以看到数据被复制到slave了.

用途:读写分离,数据备份,灾难恢复等

12.Redis主从复制

12.2 Redis主从复制过程:

配置好slave后,slave与master建立连接,然后发送sync命令.无论是第一次连接还是重新连接,master都会启动一个后台进程,将数据库快照保存到文件中,同时master主进程会开始收集新的写命令并缓存. 后台进程完成写文件后,master就发送文件给slave,slave将文件保存到硬盘上,再加载到内存中, 接着master就会把缓存的命令转发给slave,后续master将收到的写命令发送给slave. 如果master同时收到多个slave发来的同步连接命令,master只会启动一个进程来写数据库镜像, 然后发送给所有的slave.

12.Redis主从复制

12.3 Redis主从复制特点:

1. master可以拥有多个slave.
2. 多个slave可以连接同一个master外,还可以连接到其他slave.
3. 主从复制不会阻塞master,在同步数据时,master可以继续处理client请求.
4. 可以在master禁用数据持久化,注释掉master配置文件中的所有save配置,只需在slave上配置数据持久化.
5. 提高系统的伸缩性.

12 Redis主从复制

12.4 Redis主从复制速度:

官方提供了一个数据, Slave在21秒即完成了对Amazon网站 10G key set的复制.

13.Redis客户端

13.1 Redis的客户端非常丰富，几乎所有流行的语言都有客户端.

客户端列表:http://redis.io/clients

Java客户端推荐Jedis： https://github.com/xetorthio/jedis

13.Redis客户端

13.2 Jedis目前Release版本是2.0.0,支持的特性如下,一句话概括, 该有的都有了, 不该有的也有了:

- Sorting
- Connection handling
- Commands operating on any kind of values
- Commands operating on string values
- Commands operating on hashes
- Commands operating on lists
- Commands operating on sets
- Commands operating on sorted sets
- Transactions
- Pipelining
- Publish/Subscribe
- Persistence control commands
- Remote server control commands
- Connection pooling
- Sharding (MD5, MurmureHash)
- Key-tags for sharding
- Sharding with pipelining

13.Redis客户端

13.3 Jedis使用

添加Maven依赖:

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.0.0</version>
    <type>jar</type>
    <scope>compile</scope>
</dependency>
```

最简单的使用方式:

```
Jedis jedis = new Jedis("localhost");
jedis.set("foo", "bar");
String value = jedis.get("foo");
```

更多高级用法参考:https://github.com/xetorthio/jedis/wiki

14.Redis shard

14.1 目前,Redis server没有提供shard功能,只能在client端实现.

Redis有些客户端实现了shard,比如Java客户端Jedis.

Jedis使用一致性哈希算法实现shard,提供JedisPool,JedisPoolConfig,JedisSharedInfo,ShardedJedisPool等相关类来使用shared功能.

14.Redis shard

14.2 Jedis shardedJedisPool的创建例子:

```
<bean id="dataJedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
        <property name="maxActive" value="200"/>
        <property name="maxIdle" value="200"/>
        <property name="maxWait" value="1000"/>
        <property name="testOnBorrow" value="true"/>
</bean>
<bean id="dataJedisShardInfo" class="redis.clients.jedis.JedisShardInfo">
        <constructor-arg index="0" value="${redis.host}"/>
        <constructor-arg index="1" value="${redis.port}"/>
        <constructor-arg index="2" value="10000"/>
</bean>
<bean id="dataShardedJedisPool" class="redis.clients.jedis.ShardedJedisPool">
        <constructor-arg index="0" ref="dataJedisPoolConfig"/>
        <constructor-arg index="1">
                <list>
                        <ref bean="dataJedisShardInfo"/>
                </list>
        </constructor-arg>
</bean>
```

14.Redis shard

14.3 Jedis shardedJedisPool的使用:

```
ShardedJedis shardedJedis  = shardedJedisPool.getResource();
//xxoo
shardedJedisPool.returnResource(shardedJedis);
```

15.Redis cluster

当前稳定版本的redis(2.4.5)只支持简单的master-slave replication: 一个master写,多个slave读.只能通过客户端一致性哈希自己做sharding.

Redis cluster是下一阶段最重要的功能之一,会有集群的自动sharding,多节点容错等. 集群功将在3.0版本推出.

16.Redis Use In ChoiceHotels

16.1 Choice的Redis Server是一主一从,使用亚马逊AWS虚拟机, 机器配置如下:

7.5 GB memory

4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)

64-bit platform

I/O Performance: High

16.Redis Use In ChoiceHotels

16.2 现状:每天约更新30万左右的数据,现在库里有400W条纪录(400W个Key,使用的是Hash结构存储),每条数据都设有过期时间,占用了2.5G的内存.

为了提高读写性能Master关闭了Persistence功能,Slave只负责同步备份Master的数据,不对外提供服务.

另一种可选方案是用Master提供写服务,Slave提供读服务来实现读写分离.

16.Redis Use In ChoiceHotels

16.3 Master开启Save功能的影响:在dump过程中,除了磁盘有大量的IO操作以外,Redis是fork一个子进程来dump数据到硬盘,原有进程占用30%+的CPU,dump数据的子进程单独另外占用一个CPU.

Master开启Save对性能的直接影响:TPS大概减少30％.

Choice Master Redis服务器开启Save功能后的明显影响是:机器Load一直居高不下,大量请求超时. 关闭Save功能后服务器压力锐减,基本无请求超时情况发生.

16.Redis Use In ChoiceHotels

16.4 Redis开启AOF日志功能的影响:对性能有影响, 但是由于每次追加的数据量小, 所以对性能的影响相对小很多.

Choice Master Reids开启AOF功能后,机器load微升,对性能无明显影响.

16.Redis Use In ChoiceHotels

16.5 bgrewriteaof对性能的影响:为了定时减小AOF文件的大小,Redis2.4以后增加了自动的bgrewriteaof的功能,Redis会选择一个自认为负载低的情况下执行bgrewriteaof,这个重写AOF文件的过程是很影响性能的.

Choice Master开启自动bgrewriteaof功能对系统的明显影响是:高并发时段有请求超时,机器load 明显上升几倍.

16.Redis Use In ChoiceHotels

16.6 目前较好的方案是:Mater关闭Save功能,关闭AOF日志功能,以求达到性能最佳. Slave开启Save并开启AOF日志功能,并开启bgrewriteaof功能,不对外提供服务,这样Slave的负载总体上会一直略高于Master负载,但Master性能达到最好.

16.Redis Use In ChoiceHotels

16.7 总结：

从目前使用的情况来看,总体效果还是比较理想的, ChoiceHotels的价格存储使用Redis的Hash结构也非常适合,HotelCode+Date最为key, 这样的key很容易设置过期,RatePlan+RoomType作为Filed, 价格和流量是Value.

目前每天从GTA到Choice大约500w个请求,整个过程80%的请求在500毫秒内返回,基本无超时现象发生.

16.Redis Use In ChoiceHotels

16.8 Redis Master Info

redis_version:2.4.4
connected_clients:171
connected_slaves:1
used_memory_human:2.37G
used_memory_peak_human:2.46G
aof_enabled:0
expired_keys:1595004
keyspace_hits:2611419705
keyspace_misses:55827727
role:master
aof_current_size:3874203906
aof_base_size:3850549480
db0:keys=4073286,expires=4073286

17.Thank You

声明:以上信息有些数据来自其他同学的实践, 感谢所有分享的同学.

联系我
朱攀(Panos)
panos.zhu@gmail.com
panos.zhu@derbysoft.com