Redis技术

www.huawei.com

HUAWEI

目标

- 学完本课程后，您将能够:
  - 熟悉**Redis**使用的场景;
  - 掌握**Redis**系统架构;
  - 掌握**Redis**关键特性。

目录

什么是Redis

- 全称为**REmote DIctionary Service**。
- 一个开源的、**C**语言编写的、高性能**Key-Value**内存数据库。
- 支持多种数据类型，包括**string**（字符串）、**list**（链表）、**set**（集合）、**zset**（有序集合）、**hash**等。
- **2009**年由意大利开发者**Salvatore Sanfilippo**发起，最初的目的是为了提升**LLOOGG**（用户行为日志系统)性能。
- 从**2010**年**3**月**15**日起，**Redis**的开发工作由**VMware**主持。从**2013**年**5**月开始，**Redis**的开发由**Pivotal**赞助。

重要用户

Redis之父Salvatore

HUAWEI

Redis特性

特性

- 性能极高——支持超过**10**万次每秒的读写频率。
- 丰富的数据类型——支持二进制安全的**Strings**、**Lists**、**Hashes**、**Sets**以及**Ordered Sets**数据类型操作。
- 原子——所有操作都是原子性的，同时还支持对几个操作合并后的原子性执行。
- 支持发布/订阅。
- 支持持久化，通过快照和日志方式。
- 支持主从复制同步。
- 支持设置**key**过期。
- 支持无中心点方式的分布式集群。

目录

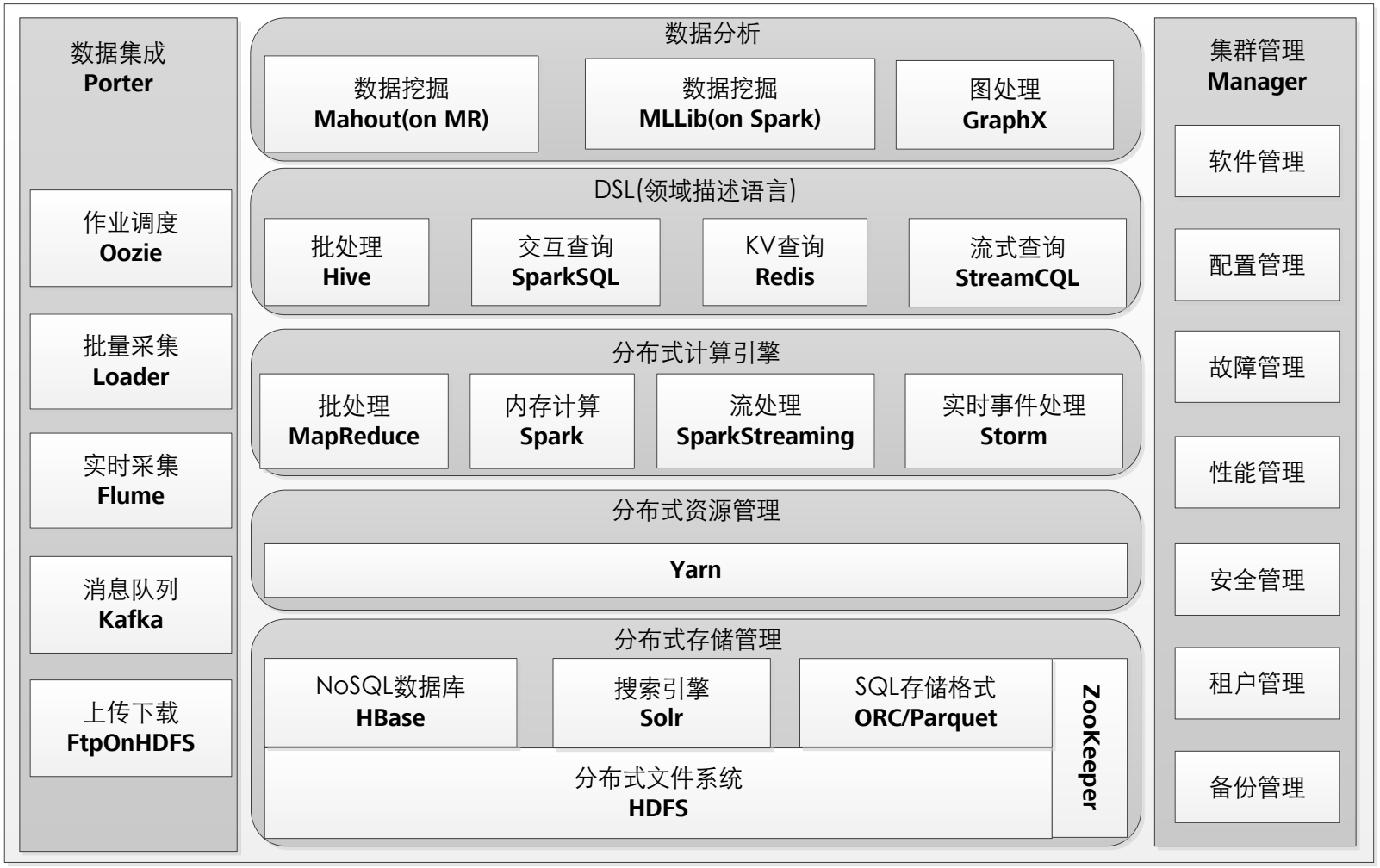
应用场景

Redis提供了灵活多变的数据结构和数据操作，主要应用于如下场景：

- 取最新**N**个数据的操作，比如典型的取某网站的最新文章。
- 排行榜应用，取**TOP N**操作。（这个需求与上面需求的不同之处在于，前面操作以时间为权重，这个是以某个条件为权重，比如按顶的次数排序。）
- 需要精准设定过期时间的应用，如网站的**Session**信息
- 计数器应用
- 构建队列系统
- 缓存

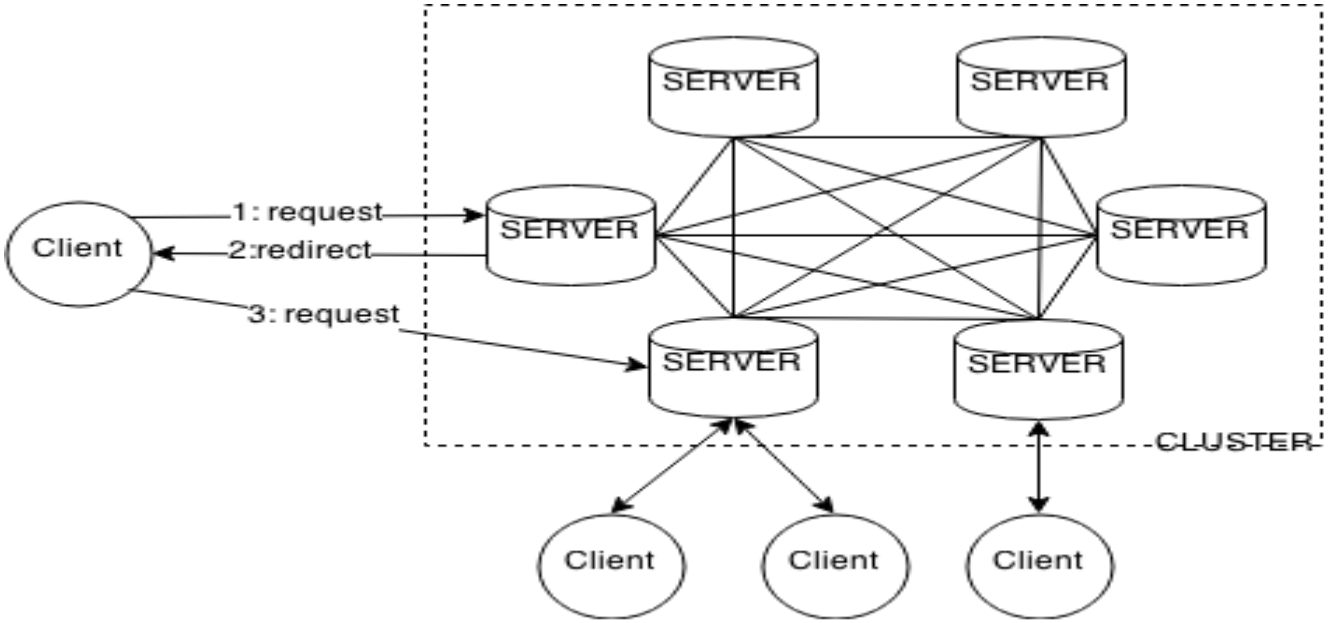
目录

Redis在FusionInsight产品的位置



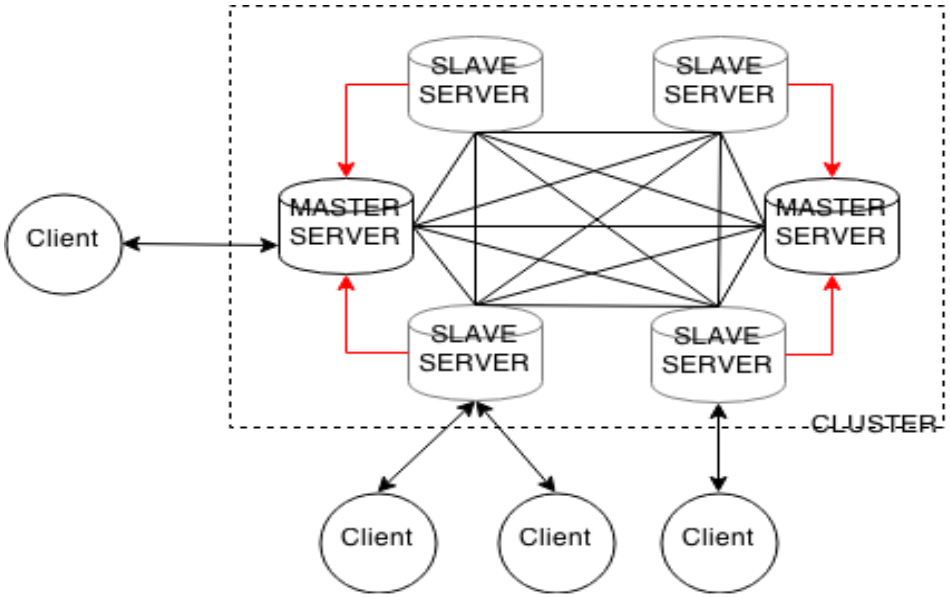
目录

基本系统架构



- 无中心自组织的结构，节点之间使用**Gossip**协议来交换节点状态信息。
- 各节点维护**Key**到**Server**地址**(IP**跟端口**)**的映射关系。
- **Client**可以向任意节点发起业务请求，节点不会转发请求，只是重定向**Client**。
- 如果在**Client**第一次请求和重定向请求之间，**Cluster**拓扑发生改变，则第二次重定向请求将被再次重定向，直到找到正确的**Server**为止。

基本系统架构



开启主从复制的**Redis**集群架构

- **Client**往**Master**端写入数据时，**Master**通过命令传播方式将命令发送至**Slave**，**Slave**也会执行相同的操作确保数据一致。
- 当**Master**宕机后，通过**Raft**选举算法从**Slave**中选举出新**Master**继续对外提供服务。**Master**恢复后以**Slave**的身份重新加入。

Redis与Memcached对比

对比项	Redis	Memcached
网络模型	单线程	多线程
持久化	支持**RDB/AOF**持久化	不支持持久化
主从复制	支持主从复制，实现故障恢复	不支持主从复制
支持的数据结构	支持**String**，**List**，**Set**，**Sorted Set**，**Hash**等丰富的数据类型	只支持简单的**key-value**
内存分配方式	现场申请内存的方式	预分配的内存池

Redis支持的数据类型

- 作为**Key-value**型数据库，**Redis**提供了键（**Key**）和键值（**Value**）的映射关系。
- **Redis**的一大特点就是支持的丰富的数据类型。主要包括如下五大类：
  - **String**——字符串
  - **List**——双向链表
  - **Set**——集合
  - **Sorted set**——有序集合
  - **Hash**——哈希表

String——字符串

- **String**是最简单的类型，可以理解为**byte[]**类型，它是“二进制安全”的，一个**key**对应一个**value**。
- **sdshdr**数据结构定义如下：

```
typedef char *sds;

struct sdshdr {
    int len;
    int free;
    char buf[];
};
```

String——字符串

- **Redis**能存储二进制安全的字符串，最大长度为**1GB**。

```
127.0.0.1:22400> set name "John Doe"
OK
127.0.0.1:22400> get name
"John Doe"
```

- **String**类型还支持批量的读写操作。

```
127.0.0.1:22400> MSET age 30 sex "male"
OK
127.0.0.1:22400> MGET age sex
1) "30"
```

二进制安全：一种主要用于字符串操作函数相关的计算机编程术语。一个二进制安全功能（函数），其本质上将操作输入作为原始的、无任何特殊格式意义的数据流。

String——字符串

- **String**类型也可以存储数字，并支持对数字的加减操作。

```
127.0.0.1:22400> INCR age
(integer) 31
127.0.0.1:22400> INCRBY age 4
(integer) 35
127.0.0.1:22400> GET age
"35"
127.0.0.1:22400> DECR age
(integer) 34
127.0.0.1:22400> DECRBY age 4
(integer) 30
127.0.0.1:22400> GET age
"30"
```

- **String**类型还支持对其部分的修改和获取操作。

```
127.0.0.1:22400> APPEND name " Mr."
(integer) 12
127.0.0.1:22400> GET name
"John Doe Mr."
127.0.0.1:22400> STRLEN name
(integer) 12
127.0.0.1:22400> SUBSTR name 0 3
"John"
```

List——双向链表

- **List**是一个双向链表结构。
- 支持双向的**Pop/Push**，一般从左端**Push**，右端**Pop**——**LPush/RPop**。
- 基于**List**可以轻松的实现最新消息排行等功能，同时还可以实现轻量级的消息队列功能。
- 支持直接查询和删除**List**中某一段元素。

List——双向链表

```
redis 127.0.0.1:6379> LINSERT students BEFORE "Captain Kirk" "Dexter Morgan"
(integer) 3
redis 127.0.0.1:6379> LRANGE students 0 2
1) "Dexter Morgan"
2) "Captain Kirk"
3) "John Doe"
redis 127.0.0.1:6379> LPUSH students "Peter Parker"
(integer) 4
redis 127.0.0.1:6379> LRANGE students 0 3
1) "Peter Parker"
2) "Dexter Morgan"
3) "Captain Kirk"
4) "John Doe"
redis 127.0.0.1:6379> LTRIM students 1 3
OK
redis 127.0.0.1:6379> LLEN students
(integer) 3
redis 127.0.0.1:6379> LRANGE students 0 2
1) "Dexter Morgan"
2) "Captain Kirk"
3) "John Doe"
redis 127.0.0.1:6379> LREM students 1 "John Doe"
(integer) 1
redis 127.0.0.1:6379> LLEN students
(integer) 1
redis 127.0.0.1:6379> LRANGE students 0 1
1) "Captain Kirk"
```

Set——集合

- **Set**是一个集合，集合就是许多不重复值的组合。
  - 利用**Redis**提供的**Set**数据结构，可以存储集合性的数据。比如在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。
  - 另外由于**Redis**非常人性化的为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能，对上面的所有集合操作，还可以使用不同的命令选择将结果返回给客户端还是存集到一个新的集合中。
- 操作命令：
  - 对某个集合添加删除元素等操作**(SAdd/SRem/SIsMember/SCard/SMove/SMembers)**。
  - 对多个集合求交并差等操作**(SInter/SInterStore/SUnion/SUnionStore/SDiff/SDiffStore)**。

Set——集合

- **Redis**能够将一系列不重复的值存储成一个集合。
- **Set**结构也支持相应的修改操作。
- **Redis**还支持对集合的子交并补等操作。

```
127.0.0.1:22400> SADD birds crow
(integer) 1
127.0.0.1:22400> SADD birds pigeon
(integer) 1
127.0.0.1:22400> SADD birds bat
(integer) 1
127.0.0.1:22400> SADD mammals dog
(integer) 1
127.0.0.1:22400> SADD mammals cat
(integer) 1
127.0.0.1:22400> SADD mammals bat
(integer) 1
127.0.0.1:22400> SMEMBERS birds
1) "bat"
2) "crow"
3) "pigeon"
127.0.0.1:22400> SMEMBERS mammals
1) "bat"
2) "cat"
3) "dog"
```

```
127.0.0.1:22400> SREM mammals cat
(integer) 1
127.0.0.1:22400> SMEMBERS mammals
1) "bat"
2) "dog"
127.0.0.1:22400> SADD mammals human
(integer) 1
127.0.0.1:22400> SMEMBERS mammals
1) "human"
2) "bat"
3) "dog"
```

```
127.0.0.1:22400> SINTER birds mammals
1) "bat"
127.0.0.1:22400> SUNION birds mammals
1) "pigeon"
2) "human"
3) "dog"
4) "bat"
5) "crow"
127.0.0.1:22400> SDIFF birds mammals
1) "crow"
2) "pigeon"
```

zSet——有序集合(Sorted Set)

- 和**Sets**相比，**Sorted Sets**是将**Set**中的元素增加了一个权重参数**score**，使得集合中的元素能够按**score**进行有序排列。
  - 这一权重属性在添加修改元素的时候可以指定，每次指定后，**zset**会自动重新按新的值调整顺序。
- 操作命令：
  - **ZRange/ZRevRange**，按排名的上下限返回元素，分别为正序与倒序。
  - **ZRangeByScore/ZRevRangeByScore**，按分数的上下限返回元素，分别为正序与倒序。
  - **ZRemRangeByRank/ZRemRangeByScore**，按排名/按分数的上下限删除元素。
  - **ZCount**，统计分数上下限之间的元素个数。
  - **ZRank/ZRevRank**，显示某个元素的正倒序的排名。
  - **ZScore/ZIncrby**，显示元素的分数/增加元素的分数。
  - **ZAdd(Add)/ZRem(Remove)/ZCard(Count)**，**ZInsertStore**（交集）**/ZUnionStore**（并集）。**Set**操作，与常规**Set**相比，少了**IsMember**和差集运算。

zSet——有序集合(Sorted Set)

```
redis 127.0.0.1:6379> ZADD days 0 mon
(integer) 1
redis 127.0.0.1:6379> ZADD days 1 tue
(integer) 1
redis 127.0.0.1:6379> ZADD days 2 wed
(integer) 1
redis 127.0.0.1:6379> ZADD days 3 thu
(integer) 1
redis 127.0.0.1:6379> ZADD days 4 fri
(integer) 1
redis 127.0.0.1:6379> ZADD days 5 sat
(integer) 1
redis 127.0.0.1:6379> ZADD days 6 sun
(integer) 1
redis 127.0.0.1:6379> ZCARD days
(integer) 7
redis 127.0.0.1:6379> ZRANGE days 0 6
1) "mon"
2) "tue"
3) "wed"
4) "thu"
5) "fri"
6) "sat"
7) "sun"
redis 127.0.0.1:6379> ZSCORE days sat
"5"
redis 127.0.0.1:6379> ZCOUNT days 3 6
(integer) 4
redis 127.0.0.1:6379> ZRANGEBYSCORE days 3 6
1) "thu"
2) "fri"
3) "sat"
4) "sun"
```

Hash——哈希表/字典表

- **Key-HashMap**结构，相比**String**类型将整个对象持久化，**Hash**将对象的各个属性存入**Map**里，可以只读取/更新对象的某些属性。
- 底层实现是**hash table**，一般操作复杂度是**O(1)**，要同时操作多个**field**时就是**O(N)**，**N**是**field**的数量。
- 操作命令：
  - **hset/hget**，向名称为**key**的**hash**中添加元素或获取元素的值
  - **hmset/hmget**，向名称为**key**的**hash**中添加多个元素或获取元素的值
  - **hincrby**，将指定**hash**中的**field**的值增加
  - **hexists**，判断指定**hash**中是否存在某个字段
  - **hdel**，删除指定**hash**中的某个字段
  - **hlen**，获取指定**hash**中的元素个数
  - **hkeys**，返回指定**hash**中的所有键
  - **hvals**，返回指定**hash**表中所有键对应的**value**
  - **hgetall**，返回**hash**中所有键（**field**）和对应的值（**value**）

Hash——哈希表/字典表

- **Redis**能够存储**key**对应的多个属性的数据（比如**user1.uname user1.passwd**）。
- **Hash**数据结构能够批量修改和获取。

```
127.0.0.1:22400> HKEYS student
1) "name"
2) "age"
3) "sex"
127.0.0.1:22400> HVALS student
1) "Ganesh"
2) "30"
3) "Male"
127.0.0.1:22400> HGETALL student
1) "name"
2) "Ganesh"
3) "age"
4) "30"
5) "sex"
6) "Male"
127.0.0.1:22400> HDEL student sex
(integer) 1
127.0.0.1:22400> HGETALL student
1) "name"
2) "Ganesh"
3) "age"
4) "30"
```

```
127.0.0.1:22400> HMSET kid name Akshi age 2 sex Female
OK
127.0.0.1:22400> HMGET kid name age sex
1) "Akshi"
2) "2"
3) "Female"
```

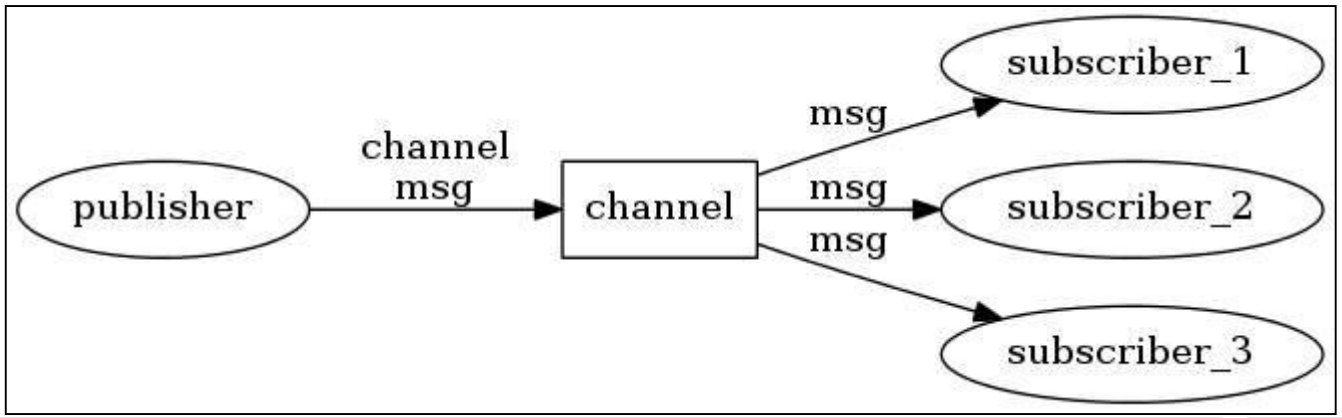
目录

Redis的发布/订阅特性

- **Redis**通过**PUBLISH**、**SUBSCRIBE**和**PSUBSCRIBE**等命令实现发布和订阅功能。
  - 可以设定对某一个**key**值进行消息发布及消息订阅，当一个**key**值上进行了消息发布后，所有订阅它的客户端都会收到相应的消息。
  - 这些命令被广泛用于构建实时消息系统，比如网络聊天室**(chatroom)**和实时广播、实时提醒等。
- 操作命令：
  - **PUBLISH**，向指定的频道发送信息。
  - **SUBSCRIBE**，订阅一个或多个频道。
  - **PSUBSCRIBE**，订阅符合给定模式的所有频道。

Redis的发布/订阅特性

- 当一个客户端通过**PUBLISH**命令向订阅者发送信息的时候，这个客户端被称为发布者（**publisher**）。
- 而当一个客户端使用**SUBSCRIBE**或者**PSUBSCRIBE**命令接收信息的时候，这个客户端被称为订阅者（**subscriber**）。
- 为了解耦发布者（**publisher**）和订阅者（**subscriber**）之间的关系，**Redis**使用了**channel**（频道）作为两者的中介——发布者将信息直接发布给**channel**，而**channel**负责将信息发送给适当的订阅者，发布者和订阅者之间。没有相互关系，也不知道对方的存在。



Redis的发布/订阅特性

- 客户端A订阅管道

```
127.0.0.1:22400> SUBSCRIBE channelone
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channelone"
3) (integer) 1
```

- 客户端B往这个管道推送信息

```
127.0.0.1:22400> PUBLISH channelone hello
(integer) 1
127.0.0.1:22400> PUBLISH channelone world
(integer) 1
```

- 然后客户端A就能获取到推送的消息

```
127.0.0.1:22400> SUBSCRIBE channelone
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channelone"
3) (integer) 1
1) "message"
2) "channelone"
3) "hello"
1) "message"
2) "channelone"
3) "world"
```

Redis的发布/订阅特性

- 客户端**A**用下面的命令订阅所有**channel**开头的信息通道

```
127.0.0.1:22400> PSUBSCRIBE channel*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "channel*"
3) (integer) 1
```

- 客户端**B**向两个**channel**推送信息

```
127.0.0.1:22400> PUBLISH channelone hello
(integer) 1
127.0.0.1:22400> PUBLISH channeltwo world
(integer) 1
```

- 然后在客户端**A**就能收到推送的信息

```
127.0.0.1:22400> PSUBSCRIBE channel*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "channel*"
3) (integer) 1
1) "pmessage"
2) "channel*"
3) "channelone"
4) "hello"
1) "pmessage"
2) "channel*"
3) "channeltwo"
4) "world"
```

Redis的事务特性

- **Redis**支持简单的事务操作，可以保证一个**client**发起的事务中的命令连续的执行，而中间不会插入其他**client**的命令。
- 一般情况下**Redis**在接收到一个**client**发来的命令后，会立即处理并返回处理结果，但是当一个**client**在一个连接中发出**multi**命令时，这个连接会进入事务上下文，该连接后续的命令并不是立即执行，而是先放到一个队列中。
- 当从此连接收到**exec**命令后，**redis**会顺序执行队列中的所有命令。并将所有命令的运行结果打包到一起返回给**client**。然后此连接结束事务上下文。
- 操作命令：
  - **MULTI**，启动一个事务，类似传统数据库的**begin transaction**。
  - **EXEC**，开始执行事务。
  - **DISCARD**，取消事务。
  - **WATCH**，监视某个键值对，在事务执行前如果该键值改变则取消事务（**CAS**）。

Redis的事务特性

- **Redis**支持自定义的命令组合。
- 可以用**DISCARD**命令来中断执行中的命令序列。

```
127.0.0.1:22400> SET counter 0
OK
127.0.0.1:22400> MULTI
OK
127.0.0.1:22400> INCR counter
QUEUED
127.0.0.1:22400> INCR counter
QUEUED
127.0.0.1:22400> INCR counter
QUEUED
127.0.0.1:22400> EXEC
1) (integer) 1
2) (integer) 2
3) (integer) 3
127.0.0.1:22400> GET counter
"3"
```

```
127.0.0.1:22400> SET newcounter 0
OK
127.0.0.1:22400> MULTI
OK
127.0.0.1:22400> INCR newcounter
QUEUED
127.0.0.1:22400> INCR newcounter
QUEUED
127.0.0.1:22400> INCR newcounter
QUEUED
127.0.0.1:22400> DISCARD
OK
127.0.0.1:22400> GET newcounter
"0"
```

Redis的持久化特性

Redis一大优势是支持持久化，并且提供了多种不同级别的持久化方式：

- **RDB**：可以在指定的时间间隔内生成数据集的时间点快照（**point-in-time snapshot**）。
- **AOF**（**Append Only File**）：记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。**AOF**文件中的命令全部以**Redis**协议的格式来保存，新命令会被追加到文件的末尾。**Redis**还可以在后台对**AOF**文件进行重写（**rewrite**），将对一个**key**的多条操作合并为一条操作，使得**AOF**文件的体积不会超出保存数据集状态所需的实际大小。
- **RDB+AOF**：可以同时使用**AOF**持久化和**RDB**持久化。在这种情况下，当**Redis**重启时，它会优先使用**AOF**文件来还原数据集，因为**AOF**文件保存的数据集通常比**RDB**文件所保存的数据集更完整。
- 无持久化：如没有数据持久化要求的，也可以关闭该功能。

Redis的持久化特性（RDB）

- 优点：
  - RDB是一个非常紧凑（compact）的文件，它保存了Redis在某个时间点上的数据集。这种文件非常适合用于进行备份。
  - RDB非常适用于灾难恢复（disaster recovery）：它只有一个文件，并且内容都非常紧凑，可以（在加密后）将它传送到别的地方进行保存。
  - RDB可以最大化Redis的性能：父进程在保存RDB文件时唯一要做的就是fork出一个子进程，然后这个子进程就会处理接下来的所有保存工作，父进程无须执行任何磁盘I/O操作。
  - RDB在恢复大数据集时的速度比AOF的恢复速度要快。
- 缺点：
  - RDB的周期快照模式在redis实例故障的时候，可能导致较多的（较长时间内的）数据丢失。（根据配置参数的不同而不同，通常是分钟级；而将RDB配置的过于频繁也显然不是一个好选择，因为这会导致频繁的fork和持续的高IO，影响性能）
  - 每次保存RDB的时候，Redis都要fork()出一个子进程，并由子进程来进行实际的持久化工作。在数据集比较庞大时，fork()可能会非常耗时，会造成服务器短时间内的业务中断。

Redis的持久化特性（AOF）

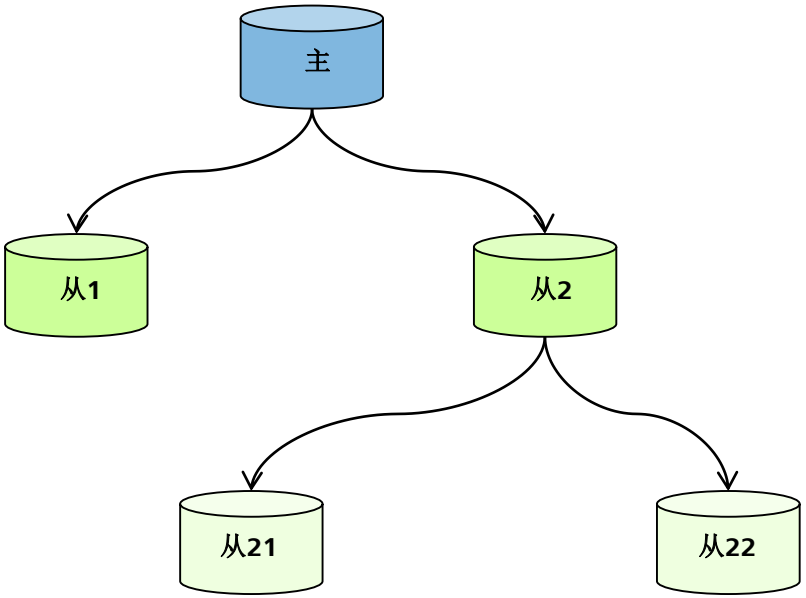
- **AOF**细分为两种模式：
  - **Everysec**：每秒写盘一次（最多丢失一秒数据，性能影响较小）。
  - **Always**：每条命令都写盘（可做到不丢失数据，性能影响较大）。
- 优点：
  - 使用**AOF**通常意味着更好的数据可靠性，可通过配置做到最多丢失一秒数据或者完全不丢失数据。
  - **AOF**还具有一个**rewrite**功能，可在后台自动对**AOF**文件进行重写，避免文件中冗余命令过多、体积过大。
  - **AOF**文件有序的保存了对数据库执行的所有写入操作，因此**AOF**文件的内容非常容易被人读懂，对文件进行分析（**parse**）也很轻松。导出（**export**）**AOF**文件也非常简单。
- 缺点：
  - 对于相同数据集来说，**AOF**的文件体积通常大于**RDB**文件。
  - 根据**fsync**的配置，**AOF**对业务的性能影响可能大于**RDB**。

Redis的持久化特性（配置建议）

- 对于数据完全作为缓存使用，不关心数据可靠性的场景：
  - 建议关闭持久化（**RDB**和**AOF**都关闭），以获得最佳性能。
- 对于数据可靠性有一定要求，但可以承担少量数据丢失的场景：
  - 建议使用**RDB**持久化,在频繁写命令的场景下，相对关闭持久化大概会有**6%**左右的性能损失；
  - 或者使用**RDB+AOF**（**everysec+fsync**模式)，相对关闭持久化大概会有**20%**左右的性能损失）。
- 对数据可靠性有极其严苛的要求，不能承受任何数据丢失的场景：
  - 建议使用**RDB+AOF(always+fsync**模式)，但此时性能影响将极其严重，大概只能达到非持久化情况下**20%**左右的性能水平，**Redis**高性能的优势将不复存在。因此强烈建议对于此种要求的场景，请重新评估是否使用**Redis**。

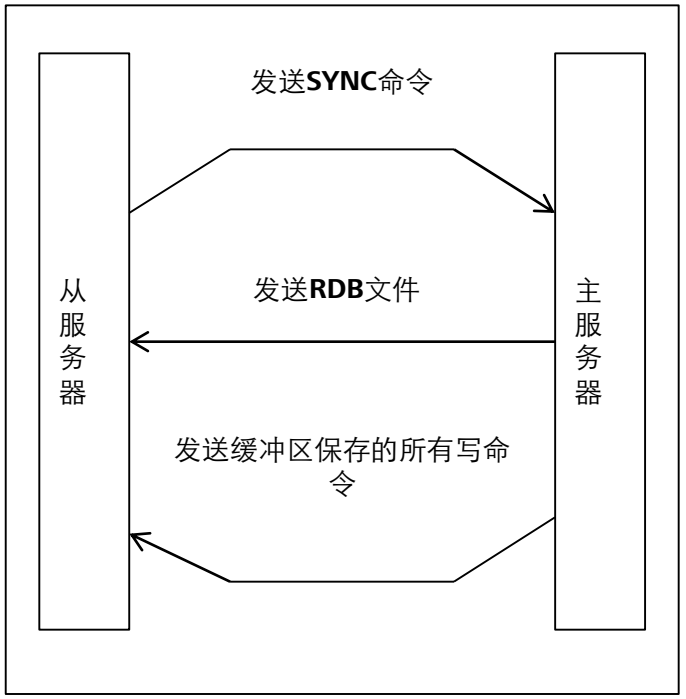
Redis的主从复制特性

- **Redis**支持将两个或两个以上的实例配置成主从模式，从而进一步提升可靠性。
- 当用户往**Master**端写入数据时，通过**Redis Sync**机制将数据文件发送至**Slave**，**Slave**也会执行相同的操作确保数据一致。
- 特点：
  - 一个主节点可以有多个从节点
  - 从节点还可以进行级联
  - 从节点默认可以进行读操作
  - 主节点上的数据复制是异步非阻塞的，只能保证最终一致性，而不能保证强一致性
  - 从节点上数据重做也是非阻塞的，即在接收新数据的同时，可以提供对老数据的读取
  - 从节点也可以执行**RDB/AOF**持久化



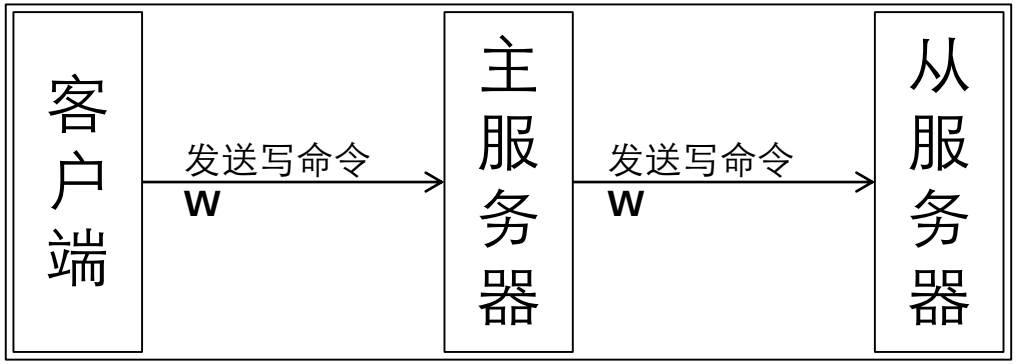
Redis的主从复制特性

- **Redis**的主从复制有两个阶段：初始复制和命令传播。
- 主从复制首先要经历初始复制阶段，就是主节点将自己的数据生成快照文件并发送给从节点，从节点进行应用，然后主从节点形成数据同步。
  - 从服务器向主服务器发送**SYNC**命令。
  - 接到**SYNC**命令的主服务器会**fork**一个子进程调用**BGSAVE**命令，创建一个**RDB**文件，同时主进程仍然可以接收读写操作，并将所有写命令记录到缓冲区中。
  - 当主服务器执行完**BGSAVE**命令时，会向从服务器发送**RDB**文件，而从服务器则会接收并载入这个文件。
  - 主服务器将缓冲区储存的所有写命令发送给从服务器执行。



Redis的主从复制特性

- 在主从节点完成初始复制之后，主节点每一条写命令都会被发送给从节点，这个操作被称为“命令传播”（**command propagate**）。
- 该操作也可称作实时复制，因为它是实时触发的，每来一条写操作，都会触发该过程。但同时它也是异步的，即主节点并不会等待从节点上接收和写入完成，因此**Redis**无法保证主从数据的强一致性。



命令传播是一个持续的过程：只要复制仍在继续，命令传播就会一直进行，使得主从服务器的状态始终保持一致。

完善的集群管理功能

- **FusionInsight HD**支持一主一从模式的**Redis**集群，系统自动计算节点上可安装的**Redis**实例个数并分配主从关系。
- 当集群需要提供大规模的处理能力时，可以一键式扩容一对或多对主从实例。在此过程中，系统会自动完成数据迁移和数据平衡，用户无需干预。
- 可以实现**Redis**的性能与告警监控。

增强的淘汰算法

- **Redis**是一个缓存系统，在其内存达到配置的最大值之后，将会触发数据的淘汰。原生**Redis**具备**LRU**、**Random**、**TTL**三种淘汰策略，但是在实际业务使用中并不能很好的达到“淘汰冷数据、留下热数据”的目的。
- **FusionInsight HD**中的**Redis**组件对淘汰算法做了增强，引入了**Smart**淘汰策略，该淘汰策略基于对**key**的热度统计进行淘汰，确保每次尽量只淘汰最冷的数据。
- 在模拟的业务测试中，**Smart**淘汰策略的热数据命中率可以始终保持在**99%**以上，热数据淘汰率最高则也只有**3%**左右（原生**LRU**策略的这两个测试数据则分别是**85%**和**35%**）。由于热数据命中率的提高，从而也使得业务请求的**TPS**得到提高。

集群管道

- **Redis-server**支持客户端发过来的管道命令，即一次接收多个命令进行处理，这样能减少网络传输时间，极大提高**Redis-server**的每秒请求处理数量。
- **Redis**社区仅提供单实例的管道模式。
- 华为客户端通过对**Redis**进行封装，使得这种模式也能应用在集群上，并且使用方式与单实例的管道模式保持一致。

目录

Redis部署基本概念

- 角色概念说明：
  - **Redis_1~32**：提供**Redis**业务的角色，主要为数据读写操作。各角色之间是对等的。且每个节点安装的角色实例数是根据节点的**CPU**核数计算得到的。
  - 单节点分区个数 计算方法为**CPU**核数减**2**，最大不超过**32**。

Redis部署规则

- 建议**Redis**服务单独部署在数据节点，网络使用**10GE**网络。
- 内存规划：
  - 若开启持久化，每个节点需要预留一半内存
  - 根据业务数据量计算**Redis**内存占用量，需考虑**2**到**3**倍的膨胀率。
- 硬盘规划：
  - 为保证集群性能，推荐每个**Redis**实例单独挂载一个硬盘；
  - 若硬盘数目有限，需多个实例共用一个硬盘，则每个硬盘上分配的**Redis**分区数不大于**3**个。

什么是Redis集群?

- **Redis**集群是一个分布式（**distributed**）、容错（**fault-tolerant**）的**Redis**实现，集群可以使用的功能是普通单机**Redis**所能使用的功能的一个子集（**subset**）。
- **Redis**集群中不存在中心（**central**）节点或者代理（**proxy**）节点，集群的其中一个主要设计目标是达到线性可扩展性（**linear scalability**）。
- **Redis**集群为了保证一致性（**consistency**）而牺牲了一部分容错性：系统会在保证对网络断线（**net split**）和节点失效（**node failure**）具有有限（**limited**）抵抗力的前提下，尽可能地保持数据的一致性。
- 集群的容错功能是通过使用主节点（**master**）和从节点（**slave**）两种角色（**role**）的节点（**node**）来实现的：
  - 主节点和从节点使用完全相同的服务器实现，它们的功能（**functionally**）也完全一样，但从节点通常仅用于替换失效的主节点。
  - 不过，如果不需要保证“先写入，后读取”操作的一致性（**read-after-write consistency)**，那么可以使用从节点来执行只读查询。

Redis集群实现的功能子集

- **Redis**集群实现了单机**Redis**中，所有处理单个数据库键的命令。
- 针对多个数据库键的复杂计算操作，比如集合的并集操作、合集操作没有被实现，那些理论上需要使用多个节点的多个数据库键才能完成的命令也没有被实现。
- **Redis**集群不像单机**Redis**那样支持多数据库功能，集群只使用默认的**0**号数据库，并且不能使用**SELECT**命令。

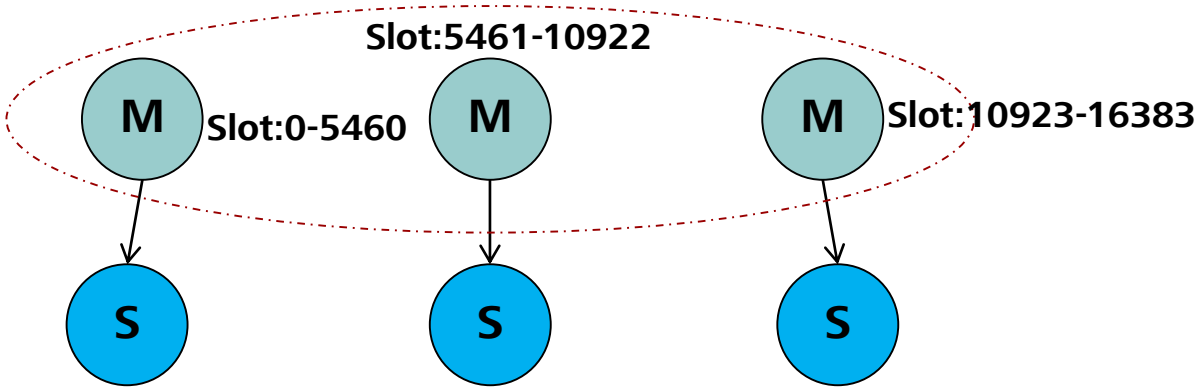
Redis集群协议中的服务器

- **Redis**集群中的节点有以下责任：
  - 持有键值对数据。
  - 记录集群的状态，包括键到正确节点的映射（**mapping keys to right nodes**）。
  - 自动发现其他节点，识别工作不正常的节点，并在有需要时，在从节点中选举出新的主节点。
- 节点之间使用**Gossip**协议来进行以下工作：
  - 传播（**propagate**）关于集群的信息，以此来发现新的节点。
  - 向其他节点发送**PING**数据包，以此来检查目标节点是否正常运作。
  - 在特定事件发生时，发送集群信息。

Redis集群协议中的客户端

- 因为集群节点不能代理（**proxy**）命令请求，所以客户端应该在节点返回**-MOVED**或者**-ASK**转向（**redirection**）错误时，自行将命令请求转发至其他节点。
- 因为客户端可以自由地向集群中的任何一个节点发送命令请求，并可以在有需要时，根据转向错误所提供的信息，将命令转发至正确的节点，所以在理论上来说，客户端是无须保存集群状态信息的。
- 客户端将键和节点之间的映射信息保存起来，可以有效地减少可能出现的转向次数，以提升命令执行的效率。

键分布模型



- **Redis**集群的键空间被分割为**16384**个槽（**slot**），集群的最大节点数量也是**16384**个。
- 键映射到槽的算法:

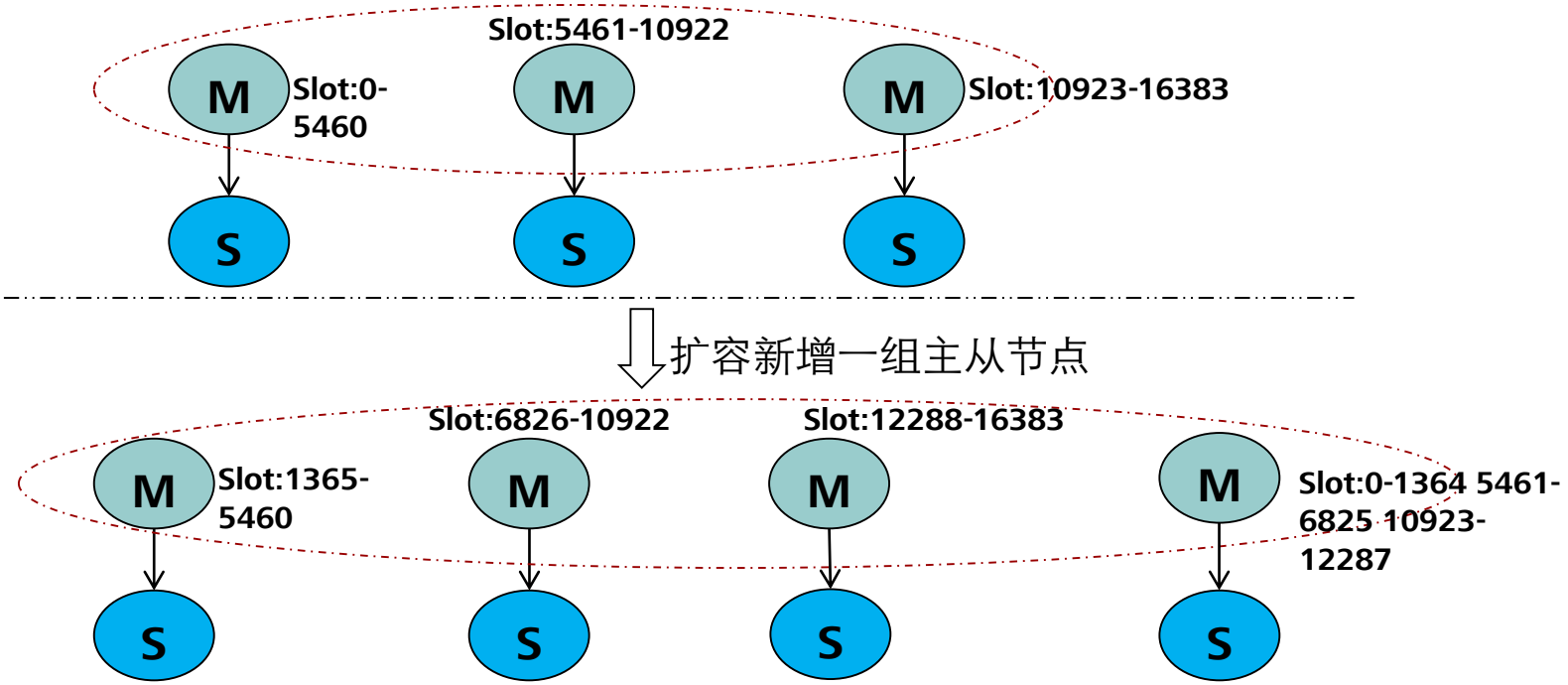
HASH_SLOT = CRC16(key) mod 16384

集群节点属性

```
$ redis-cli cluster nodes
d1861060fe6a534d42d8a19aeb36600e18785e04 :0 myself - 0 1318428930 connected 0-1364
3886e65cc906bfd9b1f7e7bde468726a052d1dae 127.0.0.1:6380 master - 1318428930
   1318428931 connected 1365-2729
d289c575dcbc4bdd2931585fd4339089e461a27d 127.0.0.1:6381 master - 1318428931
   1318428931 connected 2730-4095
```

- 在上面列出的三行信息中，从左到右的各个域分别是：节点**ID**，**IP**地址和端口号，标志（**flag**），最后发送**Ping**的时间，最后接收**Ping**的时间，连接状态，节点负责处理的槽。
- 节点**ID**用于标识集群中的每个节点。一个节点可以改变它的**IP**和端口号，而不改变节点**ID**。

集群扩容



集群扩容

- 集群扩容过程就是节点加入集群、槽位重新均衡的过程。
- 假如将哈希槽从节点**A**移到节点**B**：
  - **1.**向节点**B**发送**CLUSTER SETSLOT slot IMPORTING A**。
  - **2.**向节点**A**发送**CLUSTER SETSLOT slot MIGRATING B**。
  - **3.**向节点**A**发送**CLUSTER GETKEYSINSLOT slot count**，返回指定的哈希槽中**count**个键。
  - **4.**对上面返回的**count**个键，依次向节点**A**发送**MIGRATE target_host target_port key target_database timeout**，把指定的键从节点**A**移到节点**B**。
  - **5.**该哈希槽的所有键都已经从**A**迁移到**B**，则给**A**、**B**都发送**CLUSTER SETSLOT slot NODE B**。

集群减容

- 跟扩容相反，减容的过程是将槽位从待减掉节点迁移并均衡到其他节点、并将待减掉节点从集群中**forget**掉的过程。
- 减容的槽位迁移过程跟扩容类似。

本章总结

本章对**Redis**的概念和应用场景进行了简单介绍，然后介绍**Redis**在**FusionInsight**产品中的位置，以及**Redis**系统架构、关键特性，最后介绍了华为**FusionInsight**中**Redis**集群相关知识。

习题

- **1.Redis支持的数据类型不包括以下哪种?**

  A.Set

  B.String

  C.Array

  D.List

- **2.Redis特点有哪些?**

  A.高性能

  B.数据强一致性

  C.丰富的数据类型

  D.支持持久化

  E.支持集群模式

习题

- **3.Redis支持的持久化的方式有哪些?**

  A.DAT

  B.RDB

  C.LOG

  D.AOF

思考

- 对于性能要求不太高，但对数据一致性、可靠性要求很高要求不能丢失数据的情况下，该如何配置**Redis**?

Thank you

www.huawei.com