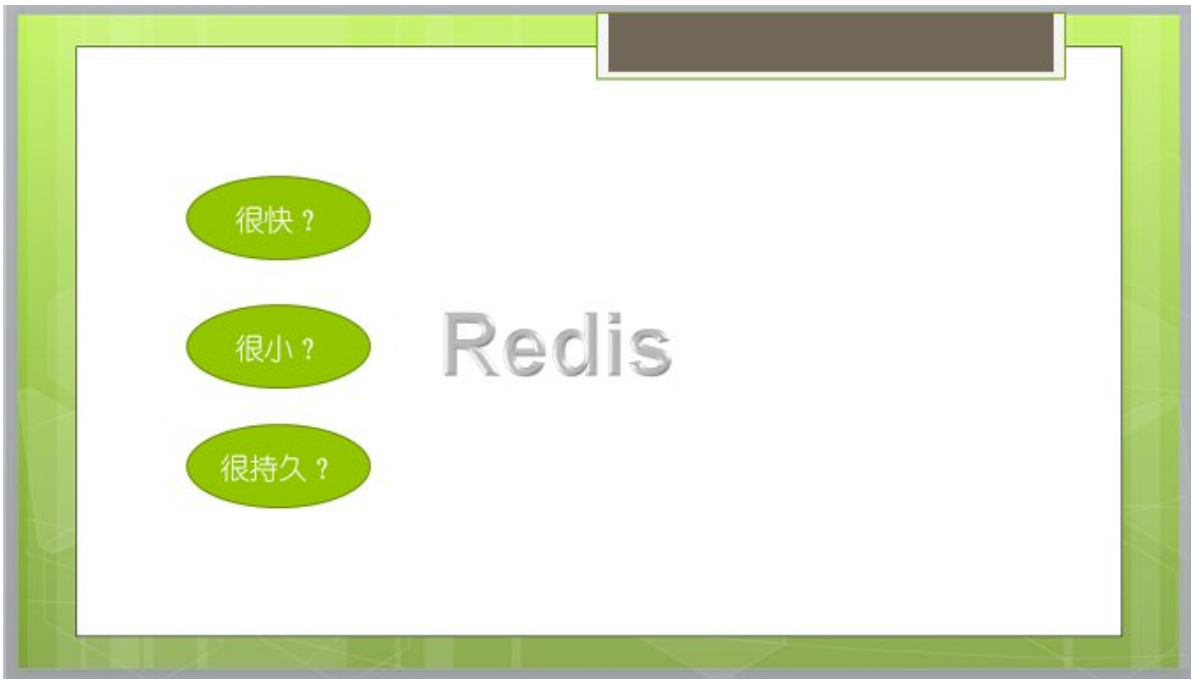
一本正经

面试官：小伙子，谈谈对Redis的看法。
我：啊，看法呀，坐着看还是躺着看。Redis很小？很快？但很持久？

面试官：一本正经的说，我怀疑你在开车，不仅开开车还搞颜色。
我：。。。

面试官：去去去，我时间有限，别瞎扯淡。回到正题，你对Redis了解有多少。
我：轻量体积小、基于内存非常快、RDB配合AOF持久化让其一样坚挺持久。

面试官：说点具体的。
我：请看正文。



正文

简介
Redis是一个开源的、高性能的、基于键值对的缓存与存储系统，通过提供多种键值数据类型来适应不同场景下的缓存与存储需求。与此同时，Redis的诸多高层级功能让其可以胜任消息队列、任务队列等不同的角色。除此之外，Redis还支持外部模块扩展，在某些特定的场景下可以作为主数据库使用。

由于内存的读写速度远快于硬盘，就算现在的固态盘思维估计也是朝着内存那个思维模式发展的，大概也许我是个外行，但是长久存储还是使用机械盘。所以Redis数据库中的所有数据都存储在内存中那是相当快的。也有一定的风险，会导致丢失数据，但配合RDB以及AOF持久化会减少风险。

一、初识Redis

1、linux下安装（Redhat7系列）

1.1、安装

此处准备的是源码包，版本不在于最新，在于稳定适用。

其余版本在官网获取，或者在其托管的平台github上获取，如下为Redis的官网下载地址。

https://redis.io/download

```
redis-6.0.8.tar.gz
#安装
tar -zxvf redis-6.0.8.tar.gz
#编译
make && make install
```

1.2、排查错误

```
make[1]: *** [server.o] 错误 1
```

```
##############################编译redis遇到的问题##############################
make[1]: *** [server.o] 错误 1
```

1.3、解决方案

1.3.1、安装依赖环境

```
yum -y install centos-release-scl
yum -y install devtoolset-9-gcc devtoolset-9-gcc-c++ devtoolset-9-binutils
```

1.3.2、加环境变量并生效

```
scl enable devtoolset-9 bash
echo "/opt/rh/devtoolset-9/enable" >> /etc/profile
```

重新读取环境变量配置文件

```
source /etc/profile
```

重新编译解决问题

```
#切换到Redis的安装目录,一般源码包安装会放在/usr/local/下面，看个人使用习惯
cd /opt/redis-6.0.8/
#编译
make && make install
```

常用基本命令练习可以参考菜鸟教程

https://www.runoob.com/redis/redis-commands.html

1.4、启动与登录

启动redis-server服务端

```
#启动redis服务
nohup /opt/redis-6.0.8/src/redis-server &
```

登录redis-cli客户端

```
#登录redis-cli
/opt/redis-6.0.8/src/redis-cli
```

测试验证，此时linux下的redis正式启动成功，下面会带来基本用法介绍。

```
ping
pong
```

```
[root@dywangk ~]# /opt/redis-6.0.8/src/redis-cli
127.0.0.1:6379> ping
PONG
```

1.5、设置密码

默认是没有开放密码设置的，需要手动开启注释掉的参数配置。

```
#编辑配置文件
vim /opt/redis-6.0.8/redis.conf

#原本的被注释掉,复制一行改成你设置的密码即可
#requirepass foobared
requirepass 123456
```

2、Windows下安装

2.1、安装

```
Redis-x64-3.2.100.zip
```

2.1.1、Windows下解压或者msi直接安装即可。

2.1.2、设置服务命令（注册为服务形式，自启）

安装服务

```
redis-server --service-install redis.windows-service.conf --loglevel verbose
```

卸载服务

```
redis-server --service-uninstall
```

2.2、启动与关闭

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

2.2.1、开启服务

```
redis-server --service-start
```

2.2.2、停止服务

```
redis-server --service-stop
```

2.3、启动redis服务

```
#同样在redis解压的或者安装的目录以管理员身份运行cmd
redis-server --service-start
```

2.4、cmd下运行测试登录

```
#在redis解压的或者安装的目录以管理员身份运行cmd
redis-cli.exe -h 127.0.0.1 -p 6379
#或者直接执行
redis-cli
#执行
redis-cli
#登录测试
ping
```

5、Windows下的管理工具rdm，是可视化界面
https://redisdesktop.com/download

二、基础知识

1、面试常问到

面试官：redis中的数据类型有哪些，能聊聊吗？

我：string（字符串类型）、hash（哈希类型）、list（列表类型）、set（集合类型）、zset（有序集合类型）、**stream（流类型）**
stream是redis5.0新增的特性支持。

面试官：嚯，小伙子有点东西啊，知道的还不少嘛，连stream流类型都知道。

我：一脸懵逼...

三、进阶

1、持久化

面试官：Redis的一些高级特性了解吗？

我：略有了解。

面试官：能具体谈谈吗？

我：飞速在大脑搜索者以前看书总结的。缓存、持久化迎面而来。

将Redis作为缓存服务器，但缓存被穿透后会对性能照成较大影响，所有缓存同时失效缓存雪崩，从而使服务无法响应。

我们希望Redis能将数据从内存中以某种形式同步到磁盘中，使之重启以后根据磁盘中的记录恢复数据。这一过程就是持久化。

面试官：知道Redis有哪几种常见的持久化方式吗？

我：Redis默认开启的RDB持久化，AOF持久化方式需要手动开启。

Redis支持两种持久化。一种是RDB方式，一种是AOF方式。前者会根据指定的规则"定时"将内存中的数据存储到硬盘上，而后者在每次执行命令后将命令本书记录下来。对于这两种持久化方式，你可以单独使用其中一种，但大多数情况下是将二者紧密结合起来。

此时的面试官一脸期待，炯炯有神的看向了我，请继续。

2、RDB方式

继续介绍，RDB采取的是快照方式，默认设置自定义快照【自动同步】，默认配置如下。

```
#   In the example below the behaviour will be to save:
#   after 900 sec (15 min) if at least 1 key changed
#   after 300 sec (5 min) if at least 10 keys changed
#   after 60 sec if at least 10000 keys changed
#
#   Note: you can disable saving completely by commenting out all "save" lines.
#
#   It is also possible to remove all the previously configured save
#   points by adding a save directive with a single empty string argument
#   like in the following example:
#
#   save ""

save 900 1
save 300 10
save 60 10000

# By default Redis will stop accepting writes if RDB snapshots are enabled
# (at least one save point) and the latest background save failed.
# This will make the user aware (in a hard way) that data is not persisting
```

RDB模式默认设置的自定义快照自动同步

同样可以手动同步

```
#不推荐在生产环境中使用
SAVE
```

```
#异步形式
BGSAVE
```

```
#基于自定义快照
FLASHALL
```

3、AOF方式

当使用Redis存储非临时数据时，一般需要打开AOF持久化来降低进程终止导致数据的丢失。AOF可以将Redis执行的每一条命令追加到硬盘文件中，着这个过程中显然会让Redis的性能打折扣，但大部分情况下这种情况可以接受。这里强调一点，使用读写较快的硬盘可以提高AOF的性能。

默认没有开启，需要手动开启AOF，当你查看redis.conf文件时也会发现appendonly配置的是no

```
appendonly yes
```

```
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"

# The fsync() call tells the Operating System to actually write data on disk
```

默认安装的Redis并没有开启AOF持久化，可以手动修改为yes

开启AOF持久化后，每次执行一条命令会会更改Redis中的数据的目录，Redis会将该命令写入磁盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置，默认的文件名是appendonly.aof，可以通过appendfilename参数修改。

```
appendfilename "appendonly.aof
```

```
# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"
```

实际上Redis也正是这样做的，每当达到一定的条件时Redis就会自动重写AOF文件，这个条件可以通过redis.conf配置文件中设置：

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

```
# This base size is compared to the current size. If the current size is
# bigger than the specified percentage, the rewrite is triggered. Also
# you need to specify a minimal size for the AOF file to be rewritten, this
# is useful to avoid rewriting the AOF file even if the percentage increase
# is reached but it is still pretty small.
#
# Specify a percentage of zero in order to disable the automatic AOF
# rewrite feature.

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

一定条件时，Redis会自动重写AOF文件

在启动时Redis会逐行执行AOF文件中的命令将硬盘中的数据加载到内存中，加载的速度相比RDB会慢一些。

虽然每次执行更改数据库内容的操作时，AOF都将命令记录在AOF文件中。但事实上，由于操作系统的缓存机制，数据并没与真正写入硬盘，而是进入了操作系统的硬盘缓存。在默认情况下，操作系统每30秒会执行一次同步操作，以便将硬盘缓存中的内容写入硬盘。

在Redis中可以通过appendfsync设置同步的时机：

```
# appendfsync always
#默认设置为everysec
appendfsync everysec
# appendfsync no
```

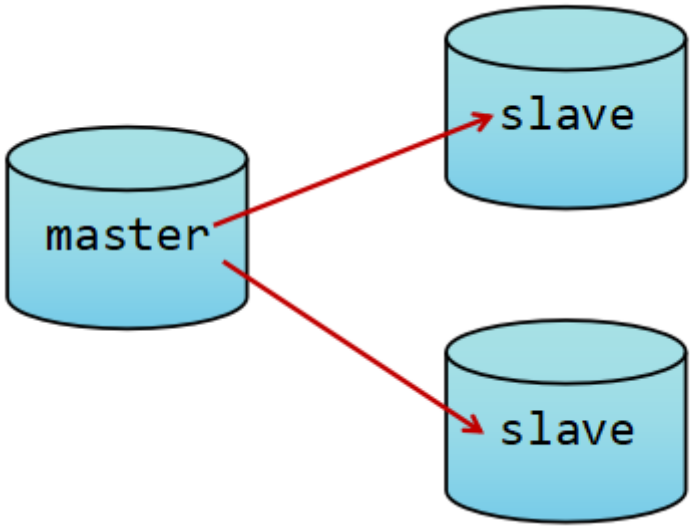
Redis允许同时开启AOF和RDB。这样既保证了数据的安全，又对进行备份等操作比较友好。此时重新启动Redis后，会使用AOF文件来恢复数据。因为AOF方式的持久化，将会丢失数据的概率降至最小化。

4、Redis复制

通过持久化功能，Redis保证了即使服务器重启的情况下也不会丢失（少部分遗失）数据。但是数据库是存储在单台服务器上的，难免不会发生各种突发情况，比如硬盘故障，服务器突然宕机等等，也会导致数据遗失。

为了尽可能的避免故障，通常做法是将数据库复制多个副本以部署在不同的服务器上。这样即使有一台出现故障，其它的服务器依旧可以提供服务。为此，Redis提供了复制（replication）功能。即实现一个数据库中的数据更新后，自动将更新的数据同步到其它数据库上。

此时熟悉MySQL的同学，是不是觉得与MySQL的主从复制很像，以开启二进制日志binlog实现同步复制。



而Redis中使用复制功能更为容易，相比MySQL而言。只需要在从库中启动时加入`slaveof`从数据库地址。

```
#在从库中配置
slaveof master_database_ip_addr
#测试,加了nohup与&是放入后台，并且输出日志到/root/目录下的nohup.out
nohup /opt/redis-6.0.8/src/redis-server --6380 --slaveof 192.168.245.147 6379 &
```

```
[root@dywangk ~]# ls
anaconda-ks.cfg  initial-setup-ks.cfg
dump.rdb         nohup.out
```

4.1、原理

复制初始化。这里主要原理是从库启动，会向主库发送SYNC命令。同时主库接收到SYNC命令后会开始在后台保存快照，即RDB持久化的过程，并将快照期间接收的命令缓存起来。当快照完成后，Redis会将快照文件和所有缓存的命令发送给从数据库。从数据库收到后，会载入快照文件并执行收到的缓存命令。

复制同步阶段会贯穿整个主从同步过程，直到主从关系终止为止。在复制的过程中快照起到了至关重要的作用，只要执行复制就会进行快照，即使关闭了RDB方式的持久化，通过删除所有save参数。

4.2、乐观复制

Redis采用了乐观复制（optimistic replication）的复制策略。容忍在一定时间内主从数据库的内容是不同的，但是两者的数据最终是会同步的。具体来讲，Redis在主从数据库之间复制数据的过程本身是异步的，这就意味着，主数据库执行完客户端请求的命令会立即将命令在主数据库的执行结果反馈给客户端，并异步的将数据同步给从库，不会等待从数据库接收到该命令在返回给客户端。

当数据至少同步给指定数量的从库时，才是可写，通过参数指定：

```
#设置最少限制3
min-slaves-to-write 3
#设置允许从数据最长失去连接时间
min-slaves-max-lag 10
```

4.3、增量复制

基于以下三点实现

- 从库会存储主库的运行ID（run id）。每个Redis运行实例均会拥有一个**唯一运行ID**，每当实例重启后，就会自动生成一个新的运行ID。类似于MySQL的从节点配置的唯一ID去识别。
- 在复制同步阶段，主库一条命令被传送到从库时，会同时把该命令存放到一个积压队列（backlog）中，记录当前积压队列中存放的命令的**偏移量范围**。
- 从库接收到主库传来的命令时，会**记录该命令的偏移量**。

4.4、注意

当主数据库崩溃时，情况略微复杂。手动通过从数据库数据库恢复主库数据时，需要严格遵循以下原则：

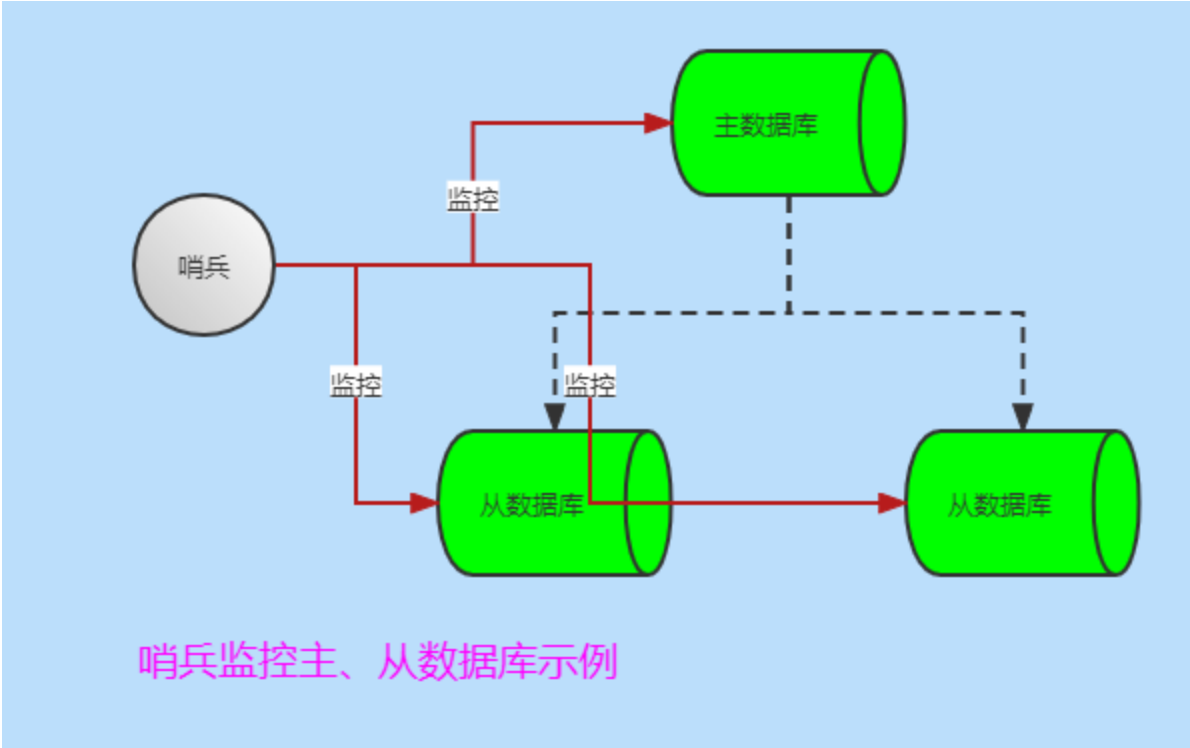
- 在从数据库中使用 `SLAVEOF NO ONE` 命令将从库提升为主库继续服务。
- 启动之前崩溃的主库，然后使用 `SLAVEOF` 命令将其设置为新的主库的从库。

注意：当开启复制且数据库关闭持久化功能时，**一定不要使用supervisor以及类似的进程管理工具令主库崩溃后重启。**同样当主库所在的服务器因故障关闭时，也要避免直接重新启动。因为当主库重启后，**没有开启持久化功能，数据库中所有数据都被清空。**此时从库依然会从主库中接收数据，**从而导致所有从库也被清空，导致数据库的持久化开了个寂寞。**

手动维护确实很麻烦，好在**Redis提供了一种自动化方案：哨兵**去实现这一过程，避免手动维护易出错的问题。

5、哨兵（sentinel）

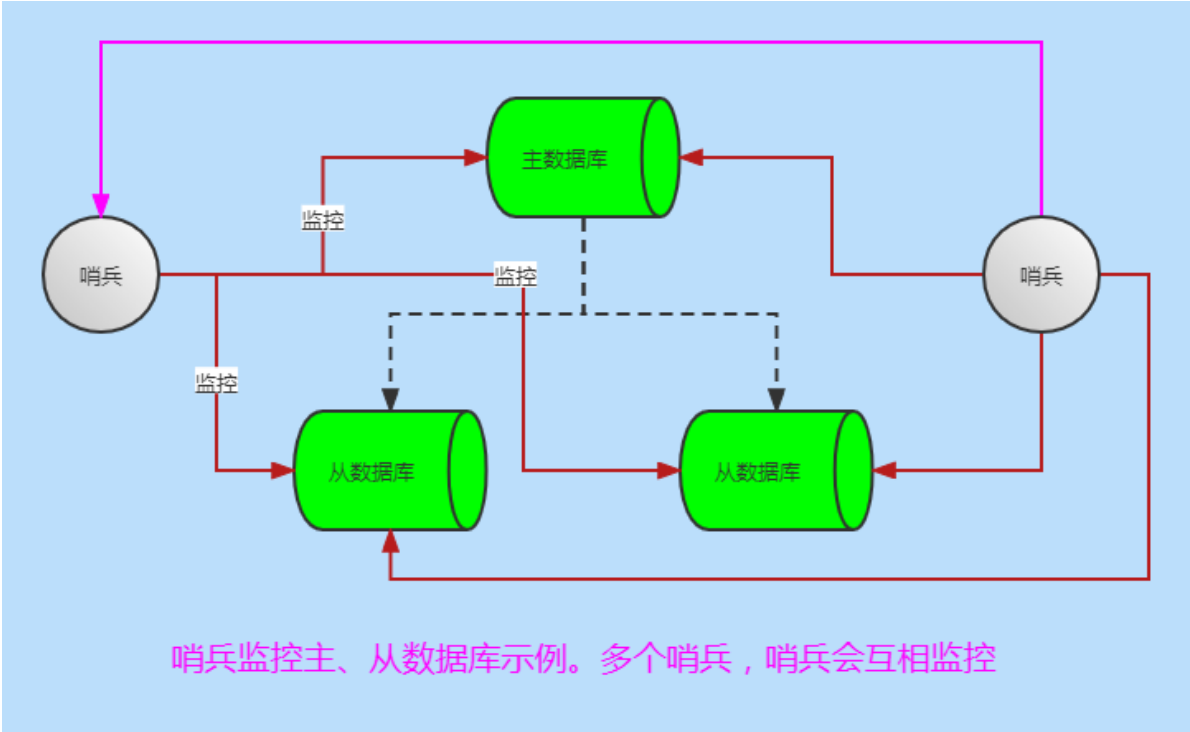
从Redis的复制历中，我们了解到在一个典型的一主多从的Redis系统中，从库在整个系统中起到了冗余备份以及读写分离的作用。当主库遇到异常中断服务后，开发人员手动将从升主时，使系统继续服务。过程相对复杂，不好实现自动化。此时可借助哨兵工具。



哨兵的作用

- 监控Redis系统运行情况
- 监控主库和从库是否正常运行
- 主库gg思密达，自动将从库升为主库，美滋滋

当然也有多个哨兵监控主从数据库模式，哨兵之间也会互相监控，如下图：



首先需要建立起一主多从的模型，然后开启配置哨兵。

```
#主库
sentinel monitor master 127.0.0.1 6379 1
#建立配置文件，例如sentinel.conf
redis-sentinel /opt/path/to/sentinel.conf
```

关于哨兵就介绍这么多，现在大脑中有印象。至少知道有那么回事，可以和美女面试官多掰扯掰扯。

6、集群（cluster）

从Redis3.0开始加入了集群这一特性。

即使使用哨兵，此时的Redis集群的每个数据库依然存有集群中的所有数据，从而导致集群的总数据存储量受限于可用内存最小的数据库节点，继而出现木桶效应。正因为Redis所有数据都是基于内存存储，问题已经很突出，尤其是当Redis作为持久化存储服务时。

有这样一种场景。就扩容来说，在客户端分片后，如果像增加更多的节点，需要对数据库进行手动迁移。迁移的过程中，为了保证数据的一致性，需要将进群暂时下线，相对比较复杂。

此时考虑到Redis很小，啊不口误，是轻量的特点。可以采用预分片（presharding）在一定程度上避免问题的出现。换句话说，就是在部署的初期，提前考虑日后的存储规模，建立足够多的实例。

从上面的理论知识来看，哨兵和集群类似，但哨兵和集群是两个独立的功能。如果要进行水平扩容，集群是不错的选择。

配置集群，开启配置文件redis.conf中的`cluster-enabled`

```
cluster-enabled yes
```

```
# Normal Redis instances can't be part of a Redis Cluster; only nodes that are
# started as cluster nodes can. In order to start a Redis instance as a
# cluster node enable the cluster support uncommenting the following:
#
# cluster-enabled yes
```

配置集群每个节点配置不同工作目录，或者修改持久化文件

```
cluster-config-file nodes-6379.conf
```

```
# Every cluster node has a cluster configuration file. This file is not
# intended to be edited by hand. It is created and updated by Redis nodes.
# Every Redis Cluster node requires a different cluster configuration file.
# Make sure that instances running in the same system do not have
# overlapping cluster configuration file names.
#
# cluster-config-file nodes-6379.conf
```

集群测试大家可以执行配置，参考其他书籍亦可，实现并不难。只要是知其原理。

四、Redis for Java

示例

```
package com.jedis;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
```

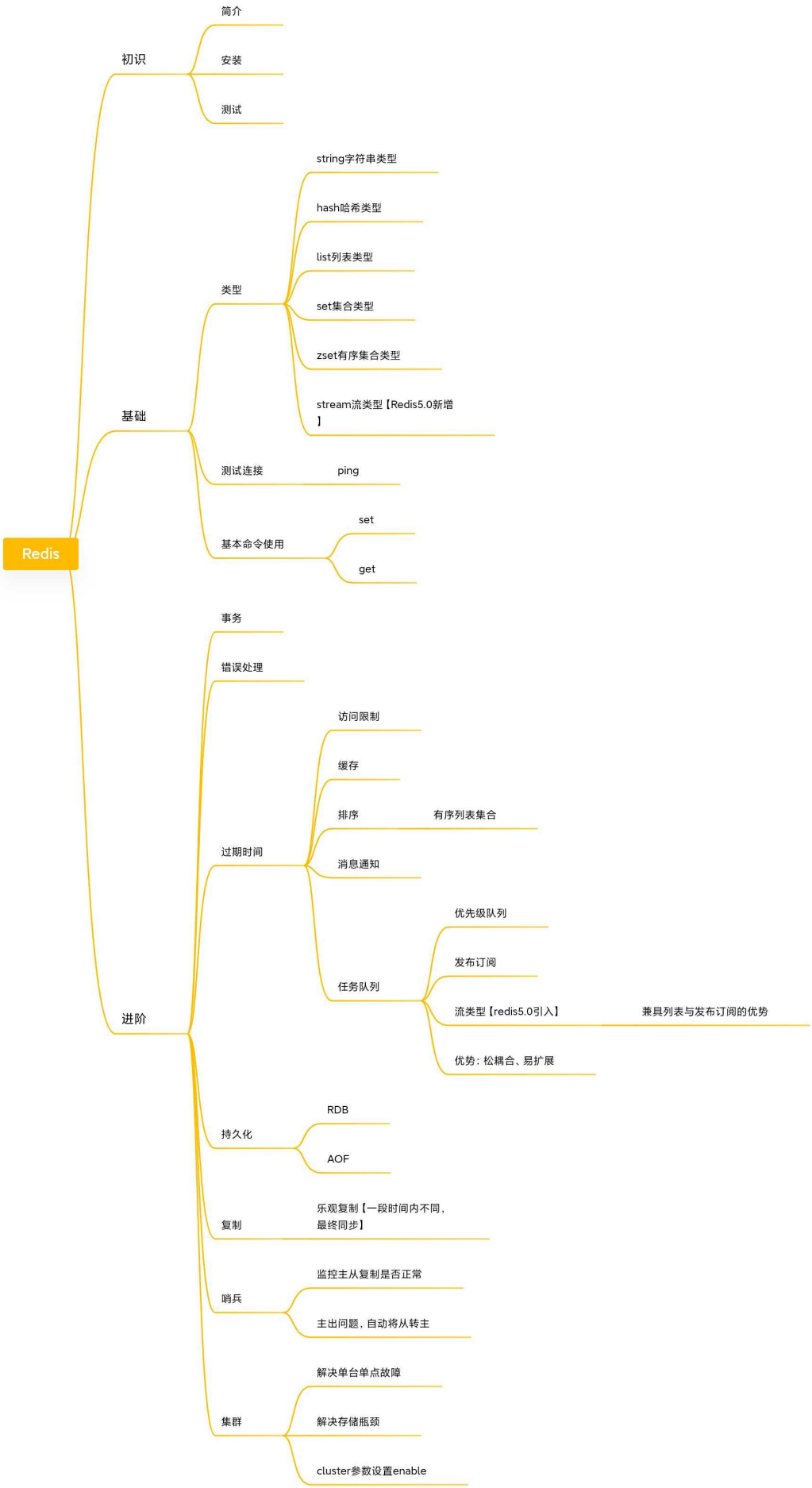
```
import redis.clients.jedis.JedisPoolConfig;

public class Test {

    @org.junit.Test
    public void demo() {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        jedis.set("name", "sky");
        String params = jedis.get("jedis");
        System.out.println(params);
        jedis.close();
    }

    @org.junit.Test
    public void config() {
        // 获取连接池的配置对象
        JedisPoolConfig config = new JedisPoolConfig();
        // 设置最大连接数
        config.setMaxTotal(30);
        // 设置最大空闲连接数
        config.setMaxIdle(10);
        // 获取连接池
        JedisPool pool = new JedisPool(config, "127.0.0.1", 6379);
        // 获得核心对象
        Jedis jedis = null;
        try {
            //通过连接池获取连接
            jedis = pool.getResource();
            //设置对象
            jedis.set("poolname", "pool");
            //获取对象
            String pools = jedis.get("poolname");
            System.out.println("values:"+pools);
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            //释放资源
            if(jedis != null){
                jedis.close();
            }
            if(pool != null){
                pool.close();
            }
        }
    }
}
```

最后放一个制作很粗超的思维导图，一时兴起，居然写了这么多。


Redis
初识
简介
安装
测试
基础
类型
string字符串类型
hash哈希类型
list列表类型
set集合类型
zset有序集合类型
stream流类型【Redis5.0新增】
测试连接
ping
基本命令使用
set
get
进阶
事务
错误处理
过期时间
访问限制
缓存
排序
有序列表集合
消息通知
任务队列
优先级队列
发布订阅
流类型【redis5.0引入】
兼具列表与发布订阅的优势
优势：松耦合、易扩展
持久化
RDB
AOF
复制
乐观复制【一段时间内不同，最终同步】
哨兵
监控主从复制是否正常
主出问题，自动将从转主
集群
解决单台单点故障
解决存储瓶颈
cluster参数设置enable

by 龙腾万里sky 原创不易，白嫖有瘾