# Redis:

工具包：

redis-4.0.1.tar.gz redis-4.0.0.gem gcc-g++

## 单例模式安装配置步骤：

1. 解压redis-4.0.1.tar.gz：

tar -zxvf redis-4.0.1.tar.gz

1. 打开解压后的文件夹：

cd redis-4.0.1

1. 编译redis

make

1. 安装redis：

make PREFIX=(指定安装路径)/usr/local/redis install

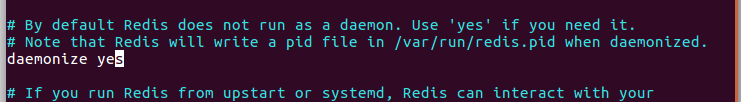
1. 拷贝一份配置文件至安装文件夹下的bin目录下

cp redis.conf /usr/local/redis/bin/

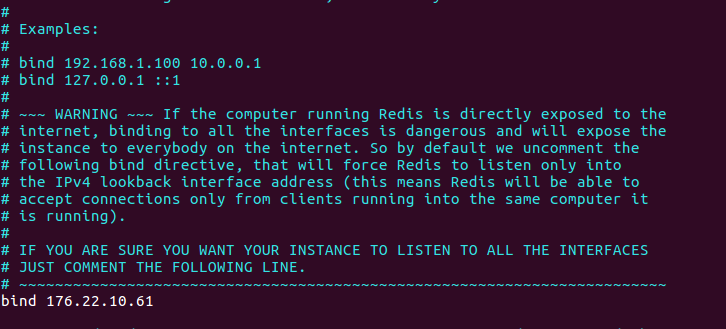
1. 修改为后台启动模式

vim /usr/local/redis/bin/redis.conf

修改daemonize为yes



修改绑定启动ip地址，默认为127.0.0.1

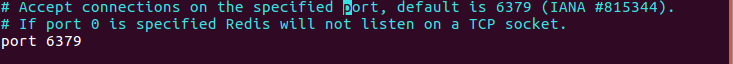


7、启动redis

cd /usr/local/redis/conf

./redis-server

redis默认端口为6379，若想修改端口只需修改redis.conf文件中的port



## redis集群安装配置（伪分布式安装）：

1. 前6步骤与单例安装相同
2. 安装ruby、rubygems、redis-4.0.0.gem

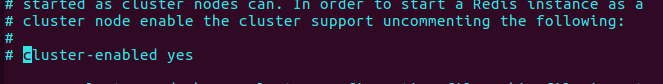
sudo apt-get install ruby

sudo apt-get install rubygems

sudo gem install redis-4.0.0.gem

3、取消配置文件中cluster-enabled的注释，避免创建多个实例需要逐一修改

vim /usr/local/redis/bin/redis.conf



4、创建多个redis实例（至少6个）

mkdir /usr/local/redis-cluster

cp /usr/local/redis /usr/local/redis-cluster/redis01

cp /usr/local/redis /usr/local/redis-cluster/redis02

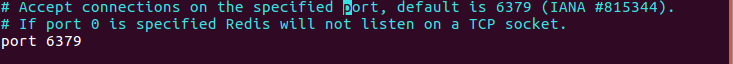
cp /usr/local/redis /usr/local/redis-cluster/redis03

cp /usr/local/redis /usr/local/redis-cluster/redis04

cp /usr/local/redis /usr/local/redis-cluster/redis05

cp /usr/local/redis /usr/local/redis-cluster/redis06

5、修改6个实例使用的端口号，在每个实例的bin目录下的redis.conf文件中



1. 拷贝redis-trib.rb至redis-cluster文件夹下,该文件位于redis解压文件夹下src文件夹中

cp /root/Downloads/redis-4.0.1/src/redis-trib.rb /usr/local/redis-cluster/

1. 启动所有实例及创建集群，使用脚本的方式启动

cd /usr/local/redis-cluster/redis01/bin

./redis-server redis.conf

cd ../..

cd redis02/bin

./redis-server redis.conf

cd ../..

cd redis03/bin

./redis-server redis.conf

cd ../..

cd redis04/bin

./redis-server redis.conf

cd ../..

cd redis05/bin

./redis-server redis.conf

cd ../..

cd redis06/bin

./redis-server redis.conf

cd ../..

./redis-trib.rb create --replicas 1 176.22.10.61:7001 176.22.10.61:7002 176.22.10.61:7003 176.22.10.61:7004 176.22.10.61:7005 176.22.10.61:7006

## redis集群安装配置：

## 在spring中整合redis--jedisclient：

### applicationContext-jedis.xml：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 连接池配置 -->

<bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<!-- 最大连接数 -->

<property name=*"maxTotal"* value=*"30"* />

<!-- 最大空闲连接数 -->

<property name=*"maxIdle"* value=*"10"* />

<!-- 每次释放连接的最大数目 -->

<property name=*"numTestsPerEvictionRun"* value=*"1024"*></property>

<!-- 释放连接的扫描间隔 -->

<property name=*"timeBetweenEvictionRunsMillis"* value=*"30000"*></property>

<!-- 连接最小空闲时间 -->

<property name=*"minEvictableIdleTimeMillis"* value=*"1800000"*></property>

<!-- 连接空闲多久后释放,当空闲时间大于该值且空闲连接大于最大空闲连接数时直接释放 -->

<property name=*"softMinEvictableIdleTimeMillis"* value=*"10000"*></property>

<!-- 获取连接时最大等待毫秒数，小于0：阻塞不确定的时间，默认：-1 -->

<property name=*"maxWaitMillis"* value=*"1500"*></property>

<!-- 获取连接的时候检查有效性,默认false -->

<property name=*"testOnBorrow"* value=*"true"*></property>

<!-- 在空闲时检查有效性,默认false -->

<property name=*"testWhileIdle"* value=*"true"*></property>

<!-- 连接耗尽时是否阻塞,false报异常,ture阻塞直到超时,默认true -->

<property name=*"blockWhenExhausted"* value=*"false"*></property>

</bean>

<!-- jedis单例配置 -->

<!-- <bean id="redisClient" class="redis.clients.jedis.JedisPool">

<constructor-arg name="host" value="176.22.10.61"></constructor-arg>

<constructor-arg name="port" value="6379"></constructor-arg>

<constructor-arg name="poolConfig" ref="redisPoolConfig"></constructor-arg>

</bean>

<bean id="jedisClient" class="com.taotao.impl.JedisClientSingle" /> -->

<!-- jedis集群配置 -->

<bean id=*"redisClient"* class=*"redis.clients.jedis.JedisCluster"*>

<constructor-arg name=*"nodes"*>

<set>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7001"*></constructor-arg>

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7002"*></constructor-arg>

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7003"*></constructor-arg>

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7004"*></constructor-arg>

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7005"*></constructor-arg>

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"176.22.10.61"*></constructor-arg>

<constructor-arg name=*"port"* value=*"7006"*></constructor-arg>

</bean>

</set>

</constructor-arg>

<constructor-arg name=*"poolConfig"* ref=*"jedisPoolConfig"*></constructor-arg>

</bean>

<bean id=*"jedisClientCluster"* class=*"com.taotao.rest.dao.impl.JedisClientCluster"*></bean>

</beans>

### JedisClient.java接口：

**package** com.taotao.rest.dao;

**public** **interface** JedisClient {

//通过key获得value

**public** String get(String key);

//添加key-value

**public** String set(String key, String value);

//添加记录

**public** String hget(String hkey, String key);

//查询记录

**public** **long** hset(String hkey, String key, String value);

//key对应的value值加

**public** **long** incr(String key);

//设置存活时间

**public** **long** expire(String key, **int** second);

//查询剩余存活时间

**public** **long** ttl(String key);

//删除记录

**public** **long** del(String key);

//删除记录

**public** **long** hdel(String hkey, String key);

}

### 单例实现类（JedisClientSingle.java）：

**public** **class** JedisClientSingle **implements** JedisClient{

@Autowired

**private** JedisPool jedisPool;

@Override

**public** String get(String key) {

Jedis jedis = jedisPool.getResource();

String string = jedis.get(key);

jedis.close();

**return** string;

}

@Override

**public** String set(String key, String value) {

Jedis jedis = jedisPool.getResource();

String string = jedis.set(key, value);

jedis.close();

**return** string;

}

@Override

**public** String hget(String hkey, String key) {

Jedis jedis = jedisPool.getResource();

String string = jedis.hget(hkey, key);

jedis.close();

**return** string;

}

@Override

**public** **long** hset(String hkey, String key, String value) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.hset(hkey, key, value);

jedis.close();

**return** result;

}

@Override

**public** **long** incr(String key) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.incr(key);

jedis.close();

**return** result;

}

@Override

**public** **long** expire(String key, **int** second) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.expire(key, second);

jedis.close();

**return** result;

}

@Override

**public** **long** ttl(String key) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.ttl(key);

jedis.close();

**return** result;

}

}

### 集群实现类（JedisClientCluster.java）：

package com.taotao.rest.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;

import com.taotao.rest.dao.JedisClient;

import redis.clients.jedis.JedisCluster;

public class JedisClientCluster implements JedisClient {

@Autowired

private JedisCluster jedisCluster;

@Override

public String get(String key) {

// TODO Auto-generated method stub

return jedisCluster.get(key);

}

@Override

public String set(String key, String value) {

// TODO Auto-generated method stub

return jedisCluster.set(key, value);

}

@Override

public String hget(String hkey, String key) {

// TODO Auto-generated method stub

return jedisCluster.hget(hkey, key);

}

@Override

public long hset(String hkey, String key, String value) {

// TODO Auto-generated method stub

return jedisCluster.hset(hkey, key, value);

}

@Override

public long incr(String key) {

// TODO Auto-generated method stub

return jedisCluster.incr(key);

}

@Override

public long expire(String key, int second) {

// TODO Auto-generated method stub

return jedisCluster.expire(key, second);

}

@Override

public long ttl(String key) {

// TODO Auto-generated method stub

return jedisCluster.ttl(key);

}

@Override

public long del(String key) {

// TODO Auto-generated method stub

return jedisCluster.del(key);

}

@Override

public long hdel(String hkey, String key) {

// TODO Auto-generated method stub

return jedisCluster.hdel(hkey, key);

}

}

## redis使用注意：

我们在添加或者使用缓存时不能影响程序的正常运行，即时redis抛出异常也要保证程序能正常运行并得到正确的结果。（后续补充）