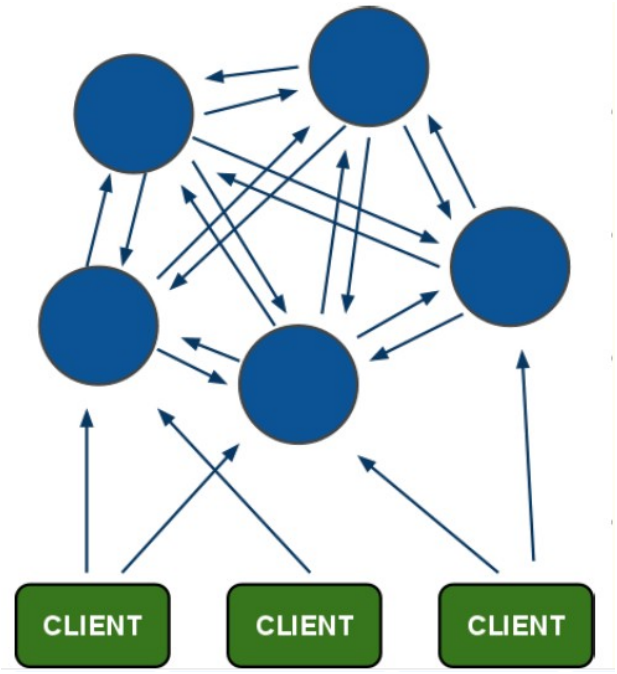
为什么要使用redis集群

单机redis为了提高网站响应速度，总是把热点数据保存在缓存中而不是直接从后端数据库中读取。单机redis故障时，redis就不能使用， 另外大型网站应用，热点数据量往往巨大,使用一台 Redis 实例无满足需求，这时就需要使用 多台 Redis （集群）作为缓存数据库。才能在用户请求时快速的进行响应。

redis集群设计架构



架构细节:

(1)所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的fail是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster把所有的物理节点映射到[0-16383]slot上,cluster 负责维护node<->slot<->value

通俗的说，redis cluster在设计的时候，就考虑到了去中心化，去中间件，也就是说，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据。

实现过程

1，创建单机redis（参考上节课）

2，单机创建节点

在单机 redis目录下创建 redis_cluster 目录；在 redis_cluster 下创建节点目录（为了方便创建和端口号一样的目录），复制redis.conf配置到节点目录下

修改配置：

port 7000 //端口 7000,7002,7003

bind 本机ip //默认ip为127.0.0.1 需要改为其他节点机器可访问的ip 否则创建集群时无法访问对应的端口，无法创建集群

daemonize yes //redis后台运行

pidfile /var/run/redis_7000.pid //pidfile文件对应 7000,7001,7002

cluster-enabled yes //开启集群 把注释#去掉

cluster-config-file nodes_7000.conf //集群的配置 配置文件首次启动自动生成 7000,7001,7002

cluster-node-timeout 15000 //请求超时 默认15秒，可自行设置

appendonly yes //aof日志开启 有需要就开启，它会每次写操作都记录一条日志

3，如果多台主机都分别执行第2步操作

4，启动及检查

启动：

./redis-server redis_cluster/7000/redis.conf

./redis-server redis_cluster/7001/redis.conf

.....

检查：

ps -ef | grep redis

netstat -tnlp | grep redis

5，使用官方提供工具创建集群

（ 使用命令前需要安装 ruby ）

redis-trib.rb 该命令是ruby程序写的，所以要下载安装ruby

yum -y install ruby ruby-devel rubygems rpm-build

//下面注释掉的废弃，不可用

/* ruby官网下载 https://www.ruby-lang.org/en/news/2018/03/28/ruby-2-3-7-released/

tar -xzvf ruby-2.3.7.tar.gz

进入 ruby-2.3.7目录

mkdir /usr/local/ruby-2.3.7

./configure --prefix=/usr/local/ruby-2.3.7

make && make install（编译安装时间较长）

*/

yum install -y centos-release-scl-rh （SCL：Software Collections SCL的设计初衷就是在不影响原有配置的前提下，让新旧软件能一起运行。 用户提供一种以方便、安全地安装和使用应用程序和运行时环境的多个（而且可能是更新的）版本的方式，同时避免把系统搞乱。）

yum install rh-ruby23 -y

scl enable rh-ruby23 bash

ruby -v

gem install redis ./gem Gem是一个管理Ruby库和程序的标准包，它通过Ruby Gem 源来查找、安装、升级和卸载软件包，非常的便捷。

使用redis提供命令创建集群

redis-trib.rb create --replicas 1 ip1:7000 ip1:7001 ip1:7002 ip2:7003 ip2:7004 ip2:7005

中间确认：Can I set the above configuration? (type 'yes' to accept): yes

[OK] All 16384 slots covered // 成功！！

链接测试

./redis-cli -h ip1 -c -p 7000 设置值，在任意其他节点取值，可以成功，说明集群成功

-c解释：连接集群结点时使用，此选项可防止moved和ask异常。

如果集群重新配置，需要删除各个单机上redis根目录下.aof .rdb .conf文件，然后重启各个节点(命令在讲课后完善，最好写成脚本)

每次set get都跳第一个原因：

Redis 集群没有并使用传统的一致性哈希来分配数据，而是采用另外一种叫做哈希槽 (hash slot)的方式来分配的。redis cluster 默认分配了 16384 个slot，当我们set一个key 时，会用CRC16算法来取模得到所属的slot，然后将这个key 分到哈希槽区间的节点上，具体算法就是：CRC16(key) % 16384。所以我们在测试的时候看到set 和 get 的时候，直接跳转到了7000端口的节点。

（详细见参考帖子）

在项目中使用集群（测试，要求jedis2.8以上版本，更换pom版本）

```
Set<HostAndPort> nodes=new HashSet<>();

        nodes.add(new HostAndPort(,));

        nodes.add(new HostAndPort(,));

        nodes.add(new HostAndPort(,));

        nodes.add(new HostAndPort(,));

        nodes.add(new HostAndPort(,));

        nodes.add(new HostAndPort(,));

        JedisCluster jedisCluster=new JedisCluster(nodes);

         // 直接使用JedisCluster对象操作redis，每次操作不需要关闭（自带连接池）。

        jedisCluster.set("aaa","123");

        String s=jedisCluster.get("aaa");
```

```
        System.out.println(s);
```

ssm项目中 ， spring-redis配置：

```
   <context:component-scan base-package="com.aaa.ssm.util">
</context:component-scan>
<bean  class="redis.clients.jedis.JedisCluster">
        <constructor-arg name="nodes">
                <set>
                        <!-- 这里配置集群中的任意一台节点即可 -->
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip1"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip1"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip1"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip2"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip2"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
                        <bean class="redis.clients.jedis.HostAndPort">
                                <constructor-arg name="host" value="ip3"/>
                                <constructor-arg name="port" value=""/>
                        </bean>
```

```
                </set>
        </constructor-arg>
<bean>
```

重新编写上节课JedisDao实现类

```
  @Autowired
    private JedisCluster jedisCluster;

    @Override
    public void putObject(Object key, Object value) {

jedisCluster.set(SerializeUtil.serialize(key),SerializeUtil.serialize(value));
        //jedisCluster.
       // JedisCluster.close();
    }

    @Override
    public Object removeObject(Object arg0) {
        return jedisCluster.expire(SerializeUtil.serialize(arg0),1);
    }

    @Override
    public Object getObject(Object arg0) {
        byte[] bytes = jedisCluster.get(SerializeUtil.serialize(arg0));
        return SerializeUtil.unserialize(bytes);
    }
```

参考帖子：

http://www.redis.net.cn/

https://www.cnblogs.com/abc-begin/p/8203613.html

https://blog.csdn.net/qq_39056805/article/details/80656811

https://www.cnblogs.com/wuxl360/p/5920330.html

http://www.cnblogs.com/yuanermen/p/5717885.html

https://www.cnblogs.com/carryping/p/7447823.html