# 关于数据库应用的总结

## 问题1:简单介绍下数据如何实现同步



1. 如果实现数据库的主从的同步 最基本的有2台机器 一主一从
2. 主从的二进制日志文件log默认是关闭的.需要手动的开启(vim /etc/my.cnf)
3. 需要实现从库挂载主库的操作change Master… master\_host,master\_port,master\_user,master\_password,master\_log\_file,master\_log\_pos
4. 同步的原理

4.1信息发生改变时,会实时的更新二进制日志文件.

4.2从库启动IO线程实时的读取主库的二进制日志为键.将数据写入中继日志中

4.3数据库的Sql线程读取中继日志的二进制文件.将数据同步到数据库中.

## 数据库高可用的实现

话术:

数据库的主从同步是实现数据库高可用的前提.刚开始公司由于业务需求量不高那么采用了**双机热备**的形式配置数据库.可以(压测) 800万/秒处理需求.

随着公司数据库中的内容不断的增多,查询的效率变低了,我们采用**分库分表的操作**,提升数据库的效率(Mycat)

# Spring整合redis

## 配置nginx实现反向代理

### 修改nginx配置文件

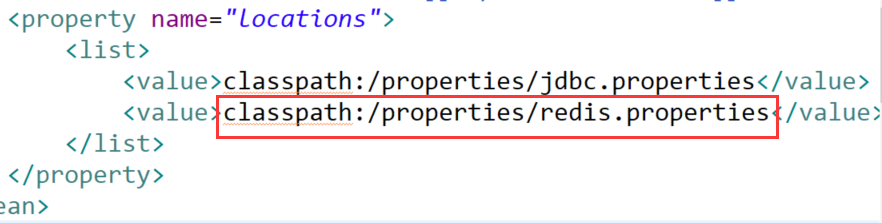


修改之后重启nginx

命令:nginx -s reload

### 配置Jedis

1. 引入redis.properties文件



1.通过bean标签配置jedis

<!--根据测试类代码实例化jedis -->

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

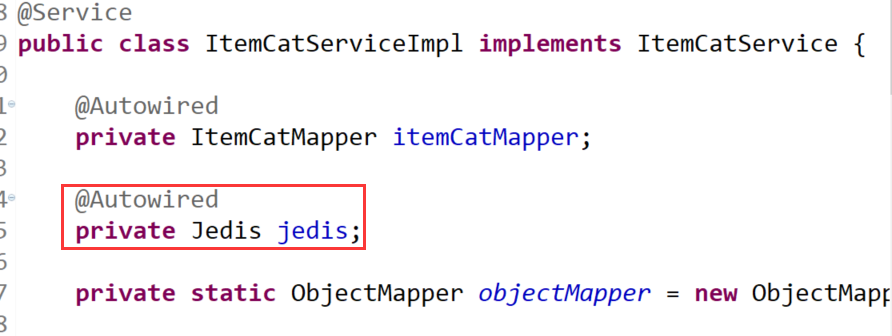
<constructor-arg name=*"host"* value=*"${redis.host}"*/>

<constructor-arg name=*"port"* value=*"${redis.port}"*/>

</bean>

### 编辑service实现缓存操作

1. 注入jedis对象



/\*\*

\* 1.通过Spring依赖注入jedis对象

\* 2.缓存实现的步骤:

\* 2.1 定义redis的key值 key = ITEM\_CAT\_0

\* 2.2 通过jedis.get(key) 判断返回数据是否为null

\* 2.3 如果数据为null,这时访问数据库进行查询.将List

\* 集合转化为JSON数据保存到redis中

\* 2.4如果数据不为空,将JSON转化为List集合对象,之后返回

\*/

@Override

**public** List<ItemCat> findItemCatList(Long parentId) {

//1.定义redis的key值

String key = "ITEM\_CAT\_"+parentId;

//2.从缓存中获取数据

String jsonData = jedis.get(key);

**try** {

//3.判断缓存中是否有数据

**if**(StringUtils.*isEmpty*(jsonData)){

//表示redis中没有缓存

ItemCat itemCat = **new** ItemCat();

itemCat.setParentId(parentId);

itemCat.setStatus(1);//查询正常的数据

List<ItemCat> itemCatList =

itemCatMapper.select(itemCat);

//将查询的数据保存到redis中

String itemCatJSON =

*objectMapper*.writeValueAsString(itemCatList);

jedis.set(key, itemCatJSON);

**return** itemCatList;

}**else**{

//缓存中有该数据 需要将JSON数据转化为List集合

//[{itemCat对象},{itemCat对象},{}]

ItemCat[] itemCats =

*objectMapper*.readValue(jsonData, ItemCat[].**class**);

List<ItemCat> itemCatList =

Arrays.*asList*(itemCats);

**return** itemCatList;

}

} **catch** (Exception e) {

e.printStackTrace();

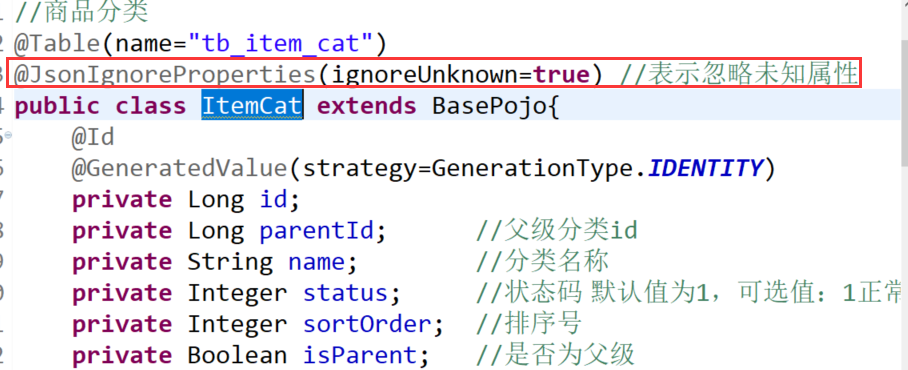
}

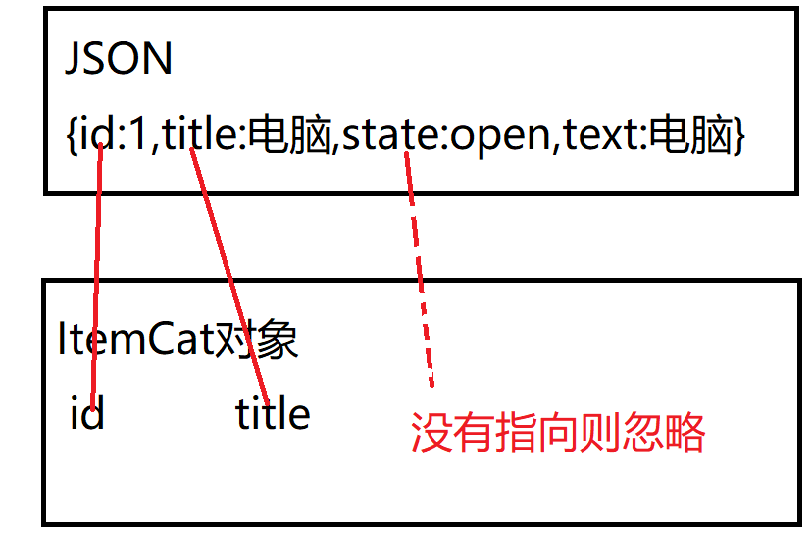
**return** **null**;

}

### 编译pojo对象

说明:因为JSON对象转化为会有多有的属性例如state/text等属性.当JSON数据转化为对象时会报错.





## 单台测试高级

### 添加redis.properties文件

redis.maxTotal=1000

redis.maxIdle=100

redis.testOnBorrow=true

redis.host=192.168.126.151

redis.port=6379

### 添加配置文件

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

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

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

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

xmlns:util="http://www.springframework.org/schema/util"

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">

<!--根据测试类代码实例化jedis -->

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

<constructor-arg name="host" value="${redis.host}"/>

<constructor-arg name="port" value="${redis.port}"/>

</bean> -->

<!--测试redis单台的高级配置 -->

<!--每次都创建jedis对象性能较低 创建jedis线程池 -->

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

<!--定义连接总数 -->

<property name="maxTotal" value="${redis.maxTotal}"/>

<!--定义最大闲置资源 -->

<property name="maxIdle" value="${redis.maxIdle}"/>

<!--定义是否自动连接测试 测试当前连接是否可用,如果有问题,则获取新的连接-->

<property name="testOnBorrow" value="${redis.testOnBorrow}"/>

</bean>

<!--定义jedis工厂模式 -->

<bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory" >

<!--定义远程主机 -->

<property name="hostName" value="${redis.host}"/>

<!--定义端口号 -->

<property name="port" value="${redis.port}"/>

<!--引入配置文件 -->

<property name="poolConfig" ref="poolConfig"/>

</bean>

<!--定义redisTemplate方便操作redis -->

<bean id="stringRedisTemplate" class="org.springframework.data.redis.core.StringRedisTemplate">

<property name="connectionFactory" ref="jedisConnectionFactory"/>

</bean>

</beans>

### 编辑Redis工具类

@Service

**public** **class** RedisService {

/\*\*

\* 说明: 在工具类中需要将自动注入的属性添加

\* **@Autowired**(required=false)注解.添加该属性后,只有在

\* 程序被调用时,对象才会自动注入

\*

\* 原因:由于工具类可以被很多程序进行引用

\* 某些项目不需要使用缓存,该对象不会实例化(不会添加redis的配置文件),

\* 则启动必然报错

\* 总结:工具类代码中依赖注入添加required=false

\*/

@Autowired(required=**false**) //调用时才会自动注入

**private** StringRedisTemplate redisTemplate;

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

ValueOperations<String,String>

operations = redisTemplate.opsForValue();

operations.set(key, value);

}

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

ValueOperations<String,String>

operations = redisTemplate.opsForValue();

System.***out***.println("单台的高级测试成功!!!!");

**return** operations.get(key);

}

### 单台测试

## Redis的分片处理

### 单台redis的缺点

1.尽管redis的功能强大,单台redis也有可能出现宕机的现象.内存的数据会受影响

2.如果采用单台的redis则内存受限,所以采用多台的redis对内存进行扩容

### Redis分片的操作

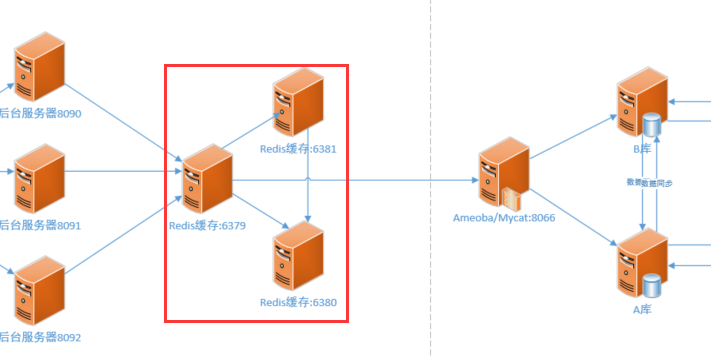
优点:

1.可以实现动态的扩容

2.如果单台redis出现问题.则数据影响较小.

缺点:

如果一台redis宕机则整合分片不能使用



## 分片的搭建过程

### 启动多台redis

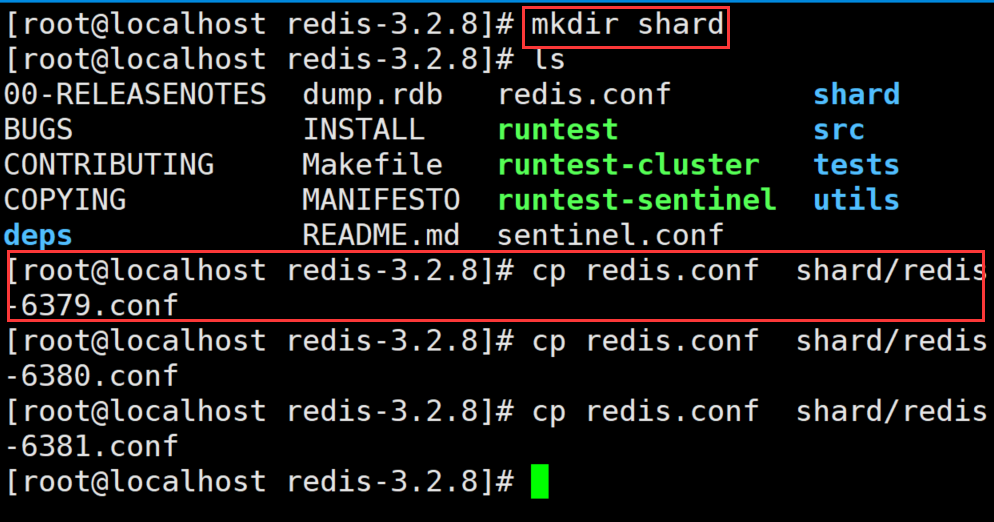
1.将原有redis关闭

redis-cli -p 6379 shutdown

1. 复制多个redis配置文件

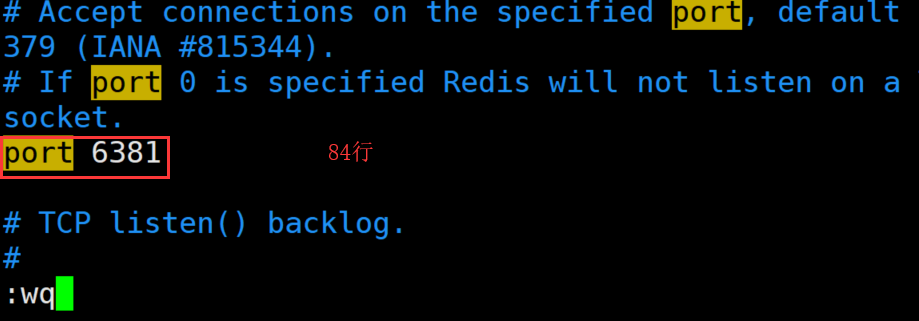
说明:在redis根目录下创建 shard文件夹

将redis.conf文件复制到shard文件夹下

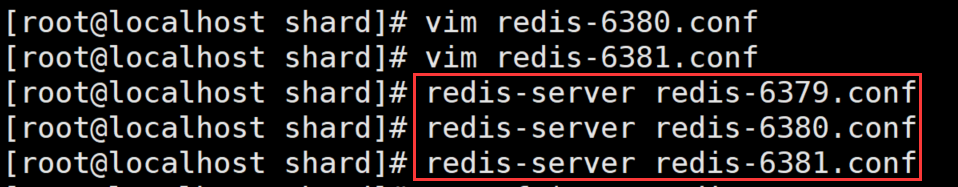


1. 修改redis端口

分别修改6380-6381的端口号



1. 启动多台redis



1. 登录redis的客户端

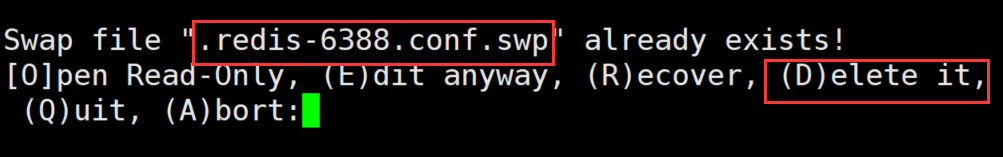
**redis-cli -p 6380**

1. **常见错误**

**说明:由于误操作将链接的线程关闭.之后该文件进入保护状态,需要人为进行设定.**

**1.D 字符表示删除之前的操作**

**2.删除保护临时文件 rm -rf .redis-6388.conf.swp**



## 分片使用

### 分片测试案例

说明:由于分片需要操作多台redis节点,使用单独的jedis不能满足要求.所以编辑如下代码.

//采用分片实现数据的缓存

@Test

**public** **void** test02(){

//1.定义池对象

JedisPoolConfig poolConfig = **new** JedisPoolConfig();

poolConfig.setMaxTotal(1000);

poolConfig.setMaxIdle(100);

poolConfig.setTestOnBorrow(**true**);

//2.定义分片的List集合

List<JedisShardInfo> infoList =

**new** ArrayList<JedisShardInfo>();

//3.添加分片的redis

infoList.add(**new** JedisShardInfo("192.168.126.151", 6379));

infoList.add(**new** JedisShardInfo("192.168.126.151", 6380));

infoList.add(**new** JedisShardInfo("192.168.126.151", 6381));

//3.定义分片的池对象

ShardedJedisPool jedisPool =

**new** ShardedJedisPool(poolConfig, infoList);

//4.获取jedis对象

ShardedJedis shardedJedis = jedisPool.getResource();

shardedJedis.set("name", "tom猫");

System.***out***.println("获取redis的数据:"+shardedJedis.get("name"));

//还池操作

jedisPool.returnResource(shardedJedis);

}

## 哈希一致性算法

1).问题1:为什么数据会落入6380???

2).问题2:数据通过get方法如何获取???

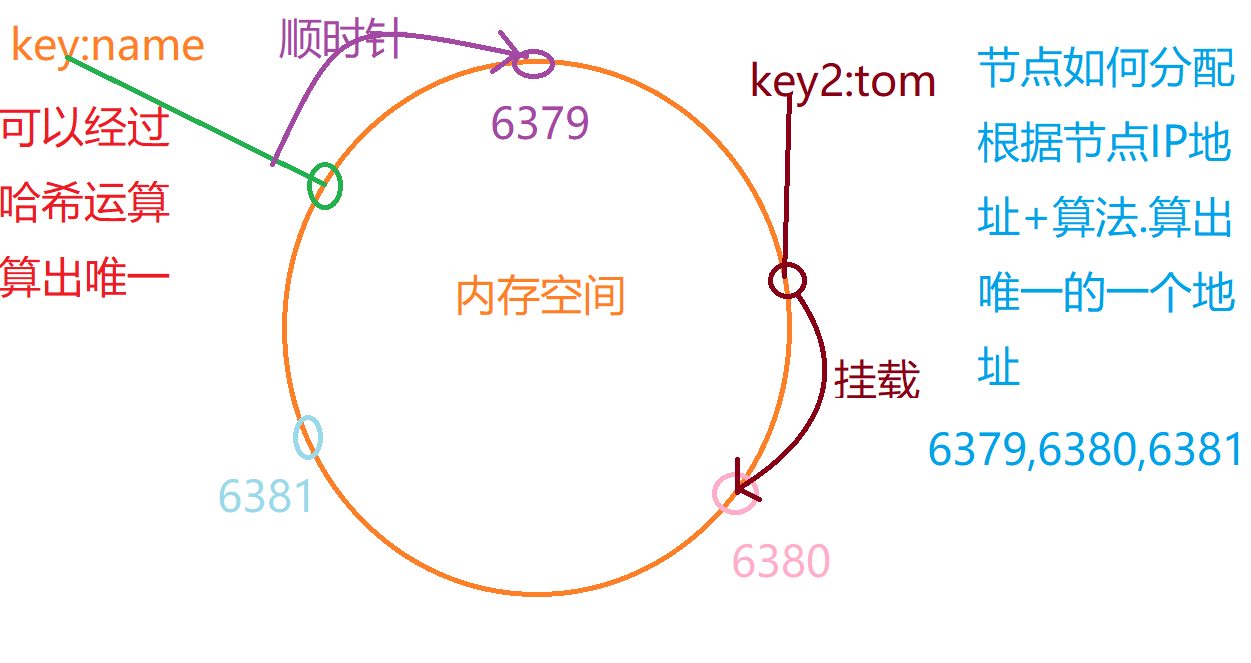
答案:因为redis中采用哈希一致性算法.

### 哈希一致性的特点

1. 分散性

说明:为什么数据会落入不同的节点中,

要素:1 key值 2.节点信息(ip地址)



总结:

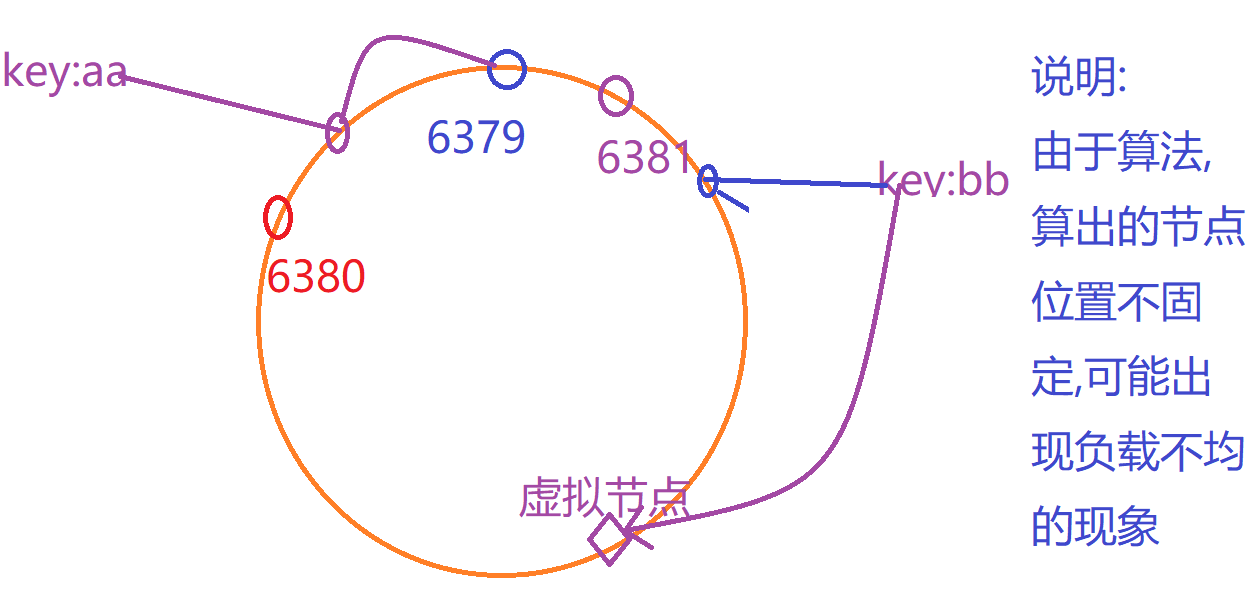
哈希一直算法中会根据key值算出唯一的一个物理地址,将数据报错该地址中.

将redis的节点信息根据IP+算法算出唯一的一个物理地址.

Key值以顺时针方向寻找最近的redis节点进行挂载.

当获取数据时.首先根据key值计算出属于哪台redis节点,之后从该节点中get(key).

1. 均衡性

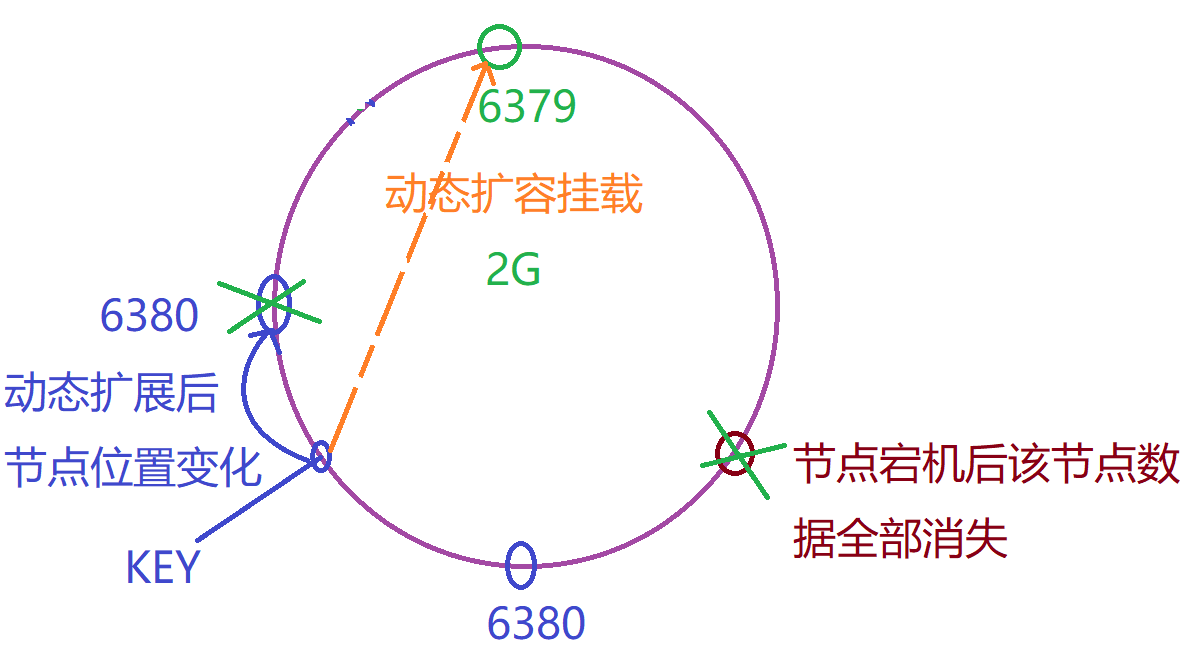


总结:

当根据哈希运算的结果出现负载不均的现象时,会为当前负载较少的节点创建虚拟节点.作用是为了分担数据的压力.保证节点所管理的信息数量尽可能维持在1/N的水平.

1. 单调性

说明:单调性可以实现动态的**数据扩容/动态数据挂载**



说明:

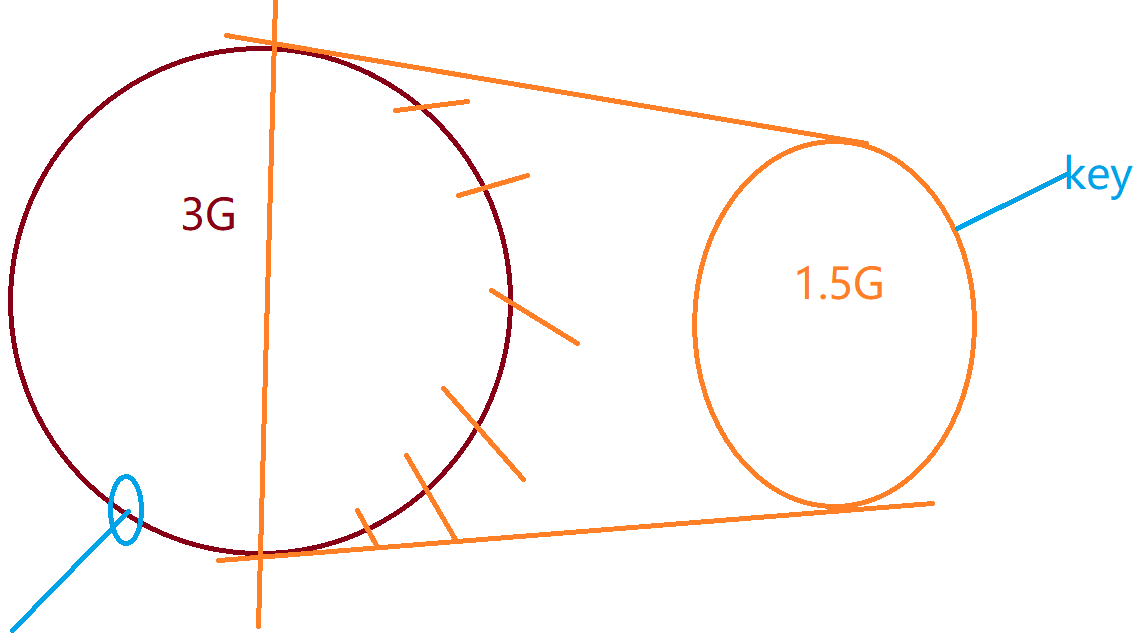
当redis中的节点出现宕机时,那么该节点中的数据全部从内存清空,那么剩余的节点会动态的进行哈希运算,剩余节点中的key值动态的实现数据挂载.

1. 负载

说明:由于**分布式**程序不能够看到**内存的全部**,则可以会产生以下问题.

1. 同一个key值可能落入不同的内存地址中
2. 同一个内存地址可能会有多个key.

哈希一致性算法要求应该尽可能的降低负载,减少出错的概率



## Spring整合分片

### 添加配置文件

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

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

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

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

xmlns:util="http://www.springframework.org/schema/util"

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">

<!--添加poolConfig文件 -->

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

<!--定义连接总数 -->

<property name="maxTotal" value="${redis.maxTotal}"/>

<!--定义最大闲置资源 -->

<property name="maxIdle" value="${redis.maxIdle}"/>

<!--定义是否自动连接测试 测试当前连接是否可用,如果有问题,则获取新的连接-->

<property name="testOnBorrow" value="${redis.testOnBorrow}"/>

</bean>

<!--定义redisInfo节点信息1 -->

<bean id="jedisShardInfo1" class="redis.clients.jedis.JedisShardInfo">

<constructor-arg name="host" value="${redis.host1}"/>

<constructor-arg name="port" value="${redis.port1}"/>

</bean>

<bean id="jedisShardInfo2" class="redis.clients.jedis.JedisShardInfo">

<constructor-arg name="host" value="${redis.host2}"/>

<constructor-arg name="port" value="${redis.port2}"/>

</bean>

<bean id="jedisShardInfo3" class="redis.clients.jedis.JedisShardInfo">

<constructor-arg name="host" value="${redis.host3}"/>

<constructor-arg name="port" value="${redis.port3}"/>

</bean>

<!--定义分片的连接池 -->

<bean id="shardedJedisPool" class="redis.clients.jedis.ShardedJedisPool">

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

<constructor-arg name="shards">

<list>

<ref bean="jedisShardInfo1"/>

<ref bean="jedisShardInfo2"/>

<ref bean="jedisShardInfo3"/>

</list>

</constructor-arg>

</bean>

</beans>

### 编辑redis.properties文件

redis.maxTotal=1000

redis.maxIdle=100

redis.testOnBorrow=true

redis.host1=192.168.126.151

redis.port1=6379

redis.host2=192.168.126.151

redis.port2=6380

redis.host3=192.168.126.151

redis.port3=6381

### 编辑工具类

//实现分片的redis操作

@Autowired

**private** ShardedJedisPool shardedJedisPool;

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

ShardedJedis jedis = shardedJedisPool.getResource();

jedis.set(key, value);

shardedJedisPool.returnResource(jedis);

}

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

ShardedJedis jedis = shardedJedisPool.getResource();

String json = jedis.get(key);

shardedJedisPool.returnResource(jedis);

System.***out***.println("分片的操作完成!!!!");

**return** json;

}