# 缓存服务

## Redis介绍

### 缓存机制

说明:缓存中的数据其实是数据库数据的备份.缓存机制其实就是有效的降低了用户访问真实物理设备的访问频次.

前提:缓存一般保存到内存中

问题:

1. 缓存的内存空间如何维护?
2. 如何保证数据一致性?
3. 缓存中的数据如何保证不丢失?
4. 如何实现缓存高可用

### Redis介绍

网址: <https://redis.io/>

Redis 是一个开源（BSD许可）的，key-value存储的.内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 [字符串（strings）](http://www.redis.cn/topics/data-types-intro.html#strings)， [散列（hashes）](http://www.redis.cn/topics/data-types-intro.html#hashes)， [列表（lists）](http://www.redis.cn/topics/data-types-intro.html#lists)， [集合（sets）](http://www.redis.cn/topics/data-types-intro.html#sets)， [有序集合（sorted sets）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) 与范围查询， [bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps)， [hyperloglogs](http://www.redis.cn/topics/data-types-intro.html" \l "hyperloglogs) 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) 索引半径查询。 Redis 内置了 [复制（replication）](http://www.redis.cn/topics/replication.html)，[LUA脚本（Lua scripting）](http://www.redis.cn/commands/eval.html)， [LRU驱动事件（LRU eviction）](http://www.redis.cn/topics/lru-cache.html)，[事务（transactions）](http://www.redis.cn/topics/transactions.html) 和不同级别的 [磁盘持久化（persistence）](http://www.redis.cn/topics/persistence.html)， 并通过 [Redis哨兵（Sentinel）](http://www.redis.cn/topics/sentinel.html)和自动 [分区（Cluster）](http://www.redis.cn/topics/cluster-tutorial.html)提供高可用性（high availability）。

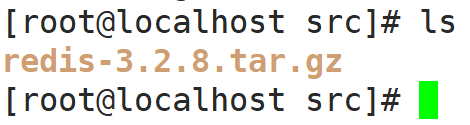
结果:**读的速度是110000次/s,写的速度是81000次/s**

### Redis下载

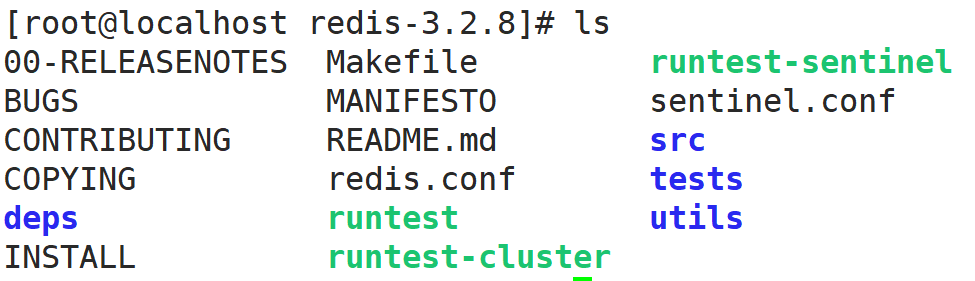


### Redis安装

1. 上传redis

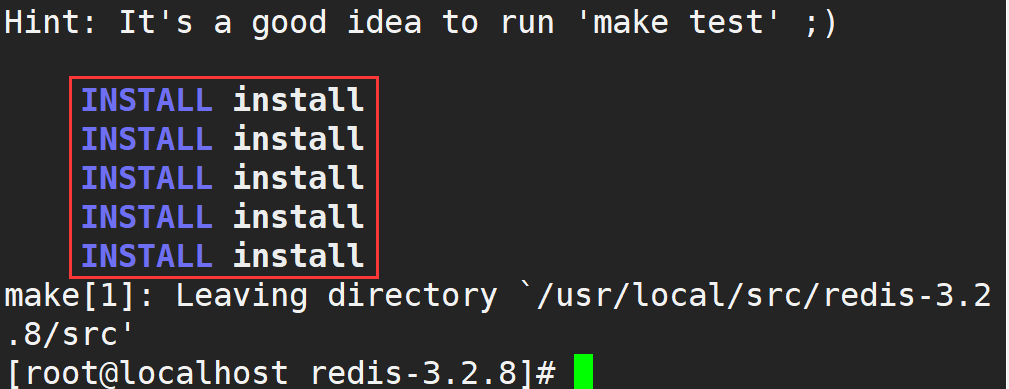


1. 解压



1. 编译/和安装redis

在redis的根目录下执行make/make install



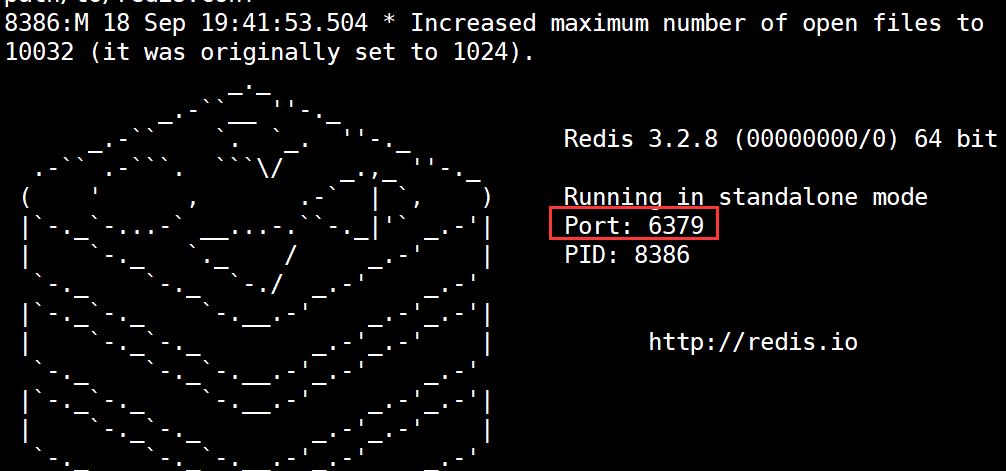
### Redis启动

1. redis-server
2. redis-server &
3. redis-server redis.conf
4. redis-cli -p 6379 shutdown

redis-cli shutdown

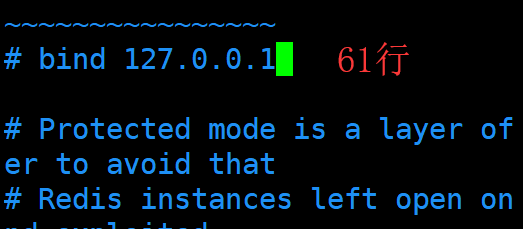
kill -9 8386

1. redis-cli -p 6379 进入控制台

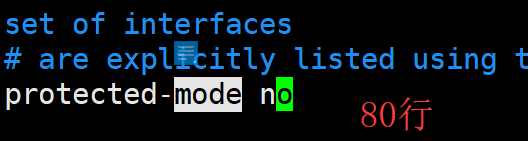


### 修改redis配置文件

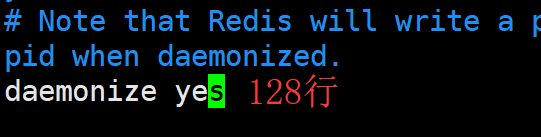
1. 关闭IP绑定



1. 关闭保护模式



1. 开启后台启动



## Redis命令

### String类型

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| set | 添加key-value | set username admin |
| get | 根据key获取数据 | get username |
| strlen | 获取key的长度 | strlen key |
| exists | 判断key是否存在 | exists name  返回1存在 0不存在 |
| del | 删除redis中的key | del key |
| Keys | 用于查询符合条件的key | keys \* 查询redis中全部的key  keys n?me 使用占位符获取数据  keys nam\* 获取nam开头的数据 |
| mset | 赋值多个key-value | mset key1 value1 key2 value2 key3 value3 |
| mget | 获取多个key的值 | mget key1 key2 |
| append | 对某个key的值进行追加 | append key value |
| type | 检查某个key的类型 | type key |
| select | 切换redis数据库 | select 0-15 redis中共有16个数据库 |
| flushdb | 清空单个数据库 | flushdb |
| flushall | 清空全部数据库 | flushall |
| incr | 自动加1 | incr key |
| decr | 自动减1 | decr key |
| incrby | 指定数值添加 | incrby 10 |
| decrby | 指定数值减 | decrby 10 |
| expire | 指定key的失效时间 单位秒 | expire key 20  key20秒后失效 |
| pexpire | 指定key的失效时间 单位毫秒 | pexpire key 2000  key 2000毫秒后失效 |
| ttl | 检查key的剩余存活时间 | ttl key |
| persist | 撤销key的失效时间 | persist key |

### Hash类型

说明:可以用散列类型保存对象和属性值

例子:User对象{id:2,name:小明,age:19}

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| hset | 为对象添加数据 | hset key field value |
| hget | 获取对象的属性值 | hget key field |
| hexists | 判断对象的属性是否存在 | HEXISTS key field  1表示存在 0表示不存在 |
| hdel | 删除hash中的属性 | hdel user field [field ...] |
| hgetall | 获取hash全部元素和值 | HGETALL key |
| hkyes | 获取hash中的所有字段 | HKEYS key |
| hlen | 获取hash中所有属性的数量 | hlen key |
| hmget | 获取hash里面指定字段的值 | hmget key field [field ...] |
| hmset | 为hash的多个字段设定值 | hmset key field value [field value ...] |
| hsetnx | 设置hash的一个字段,只有当这个字段不存在时有效 | HSETNX key field value |
| hstrlen | 获取hash中指定key的长度 | HSTRLEN key field |
| hvals | 获取hash的所有值 | HVALS user |

### List类型

说明:Redis中的List集合是双端循环列表,分别可以从左右两个方向插入数据.

List集合可以当做队列使用,也可以当做栈使用

队列:存入数据的方向和获取数据的方向相反

栈:存入数据的方向和获取数据的方向相同

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| lpush | 从队列的左边入队一个或多个元素 | LPUSH key value [value ...] |
| rpush | 从队列的右边入队一个或多个元素 | RPUSH key value [value ...] |
| lpop | 从队列的左端出队一个元素 | LPOP key |
| rpop | 从队列的右端出队一个元素 | RPOP key |
| lpushx | 当队列存在时从队列的左侧入队一个元素 | LPUSHX key value |
| rpushx | 当队列存在时从队列的右侧入队一个元素 | RPUSHx key value |
| lrange | 从列表中获取指定返回的元素 | LRANGE key start stop  Lrange key 0 -1 获取全部队列的数据 |
| lrem | 从存于 key 的列表里移除前 count 次出现的值为 value 的元素。 这个 count 参数通过下面几种方式影响这个操作：   * count > 0: 从头往尾移除值为 value 的元素。 * count < 0: 从尾往头移除值为 value 的元素。 * count = 0: 移除所有值为 value 的元素。 | LREM list -2 “hello” 会从存于 list 的列表里移除最后两个出现的 “hello”。  需要注意的是，如果list里没有存在key就会被当作空list处理，所以当 key 不存在的时候，这个命令会返回 0。 |
| Lset | 设置 index 位置的list元素的值为 value | LSET key index value |

### Redis事务命令

说明:redis中操作可以添加事务的支持.一项任务可以由多个redis命令完成,如果有一个命令失败导致入库失败时.需要实现事务回滚.

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| multi | 标记一个事务开始 | 127.0.0.1:6379> MULTI  OK |
| exec | 执行所有multi之后发的命令 | 127.0.0.1:6379> EXEC  OK |
| discard | 丢弃所有multi之后发的命令 |  |

## Redis入门案例

### 导入jar包

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>${jedis.version}</version>

</dependency>

<!--添加spring-datajar包 -->

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.4.1.RELEASE</version>

</dependency>

### 操作redis数据库

**public** **class** TestRedisString {

/\*\*

\* 步骤:

\* 1.连接Redis IP:端口

\*/

@Test

**public** **void** test01(){

Jedis jedis = **new** Jedis("192.168.126.166", 6379);

jedis.set("1805", "好好学习,早日找个异性朋友");

System.***out***.println(jedis.get("1805"));

}

//操作Hash

@Test

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

Jedis jedis = **new** Jedis("192.168.126.166", 6379);

jedis.hset("user", "id", "100");

jedis.hset("user", "name", "1805班");

Map<String,String> userMap = jedis.hgetAll("user");

System.***out***.println(userMap);

}

@Test

**public** **void** testList(){

Jedis jedis = **new** Jedis("192.168.126.166", 6379);

jedis.lpush("1805List", "1,2,2,2,2,2","2","3");

System.***out***.println(jedis.rpop("1805List"));

System.***out***.println(jedis.rpop("1805List"));

}

}

## 京淘后台缓存实现

### 什么样的数据添加缓存

说明:

1. 经常查询的数据
2. 不经常变化的数据

实际案例:

1. 商品信息
2. 省市县数据
3. 商品分类列表

## 实现商品分类缓存处理

### 实现思路

1. 将redis工具API Jedis交给spring容器管理
2. 当用户点击商品分类按钮时,应该先查询缓存,如果缓存中的数据为null,那么这时应该访问后台的数据库.
3. 将查询到的数据转化为JSON数据(objectMapper),将JSON串存入redis中.
4. 当用户再次查询时,这时缓存中有数据.将JSON串转化为java对象进行return返回.

### ObjectMapper介绍

@Test

**public** **void** javaToJSON() **throws** IOException{

ObjectMapper objectMapper = **new** ObjectMapper();

User user = **new** User();

user.setId(1);

user.setName("tomcat");

user.setAge(10);

String userJSON =

objectMapper.writeValueAsString(user);

System.***out***.println(userJSON);

//将json串转化为java对象

User user1 = objectMapper.readValue(userJSON, User.**class**);

System.***out***.println(user1.toString());

}

@Test

**public** **void** listToJSON() **throws** IOException{

ObjectMapper objectMapper = **new** ObjectMapper();

List<String> strList = **new** ArrayList<>();

strList.add("a");

strList.add("b");

strList.add("c");

strList.add("d");

strList.add("e");

String listJSON =

objectMapper.writeValueAsString(strList);

System.***out***.println(listJSON);

//将JSON转化为List集合

List<String> jsonList =

objectMapper.readValue(listJSON, List.**class**);

System.***out***.println(jsonList);

}

@Test

**public** **void** ListUserToJSON() **throws** IOException{

ObjectMapper objectMapper = **new** ObjectMapper();

User user1 = **new** User();

user1.setId(1);

user1.setName("tomcat");

user1.setAge(10);

User user2 = **new** User();

user2.setId(1);

user2.setName("tomcat");

user2.setAge(10);

List<User> userList = **new** ArrayList<>();

userList.add(user1);

userList.add(user2);

String userListJSON =

objectMapper.writeValueAsString(userList);

System.***out***.println(userListJSON);

//将userListJSON转化为java对象

ArrayList<User> arrayUserList =

objectMapper.readValue(userListJSON, ArrayList.**class**);

System.***out***.println(arrayUserList);

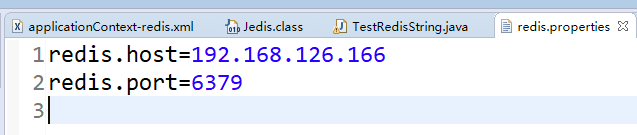
User[] users = objectMapper.readValue(userListJSON, User[].**class**);

List<User> arraysUser = Arrays.*asList*(users);

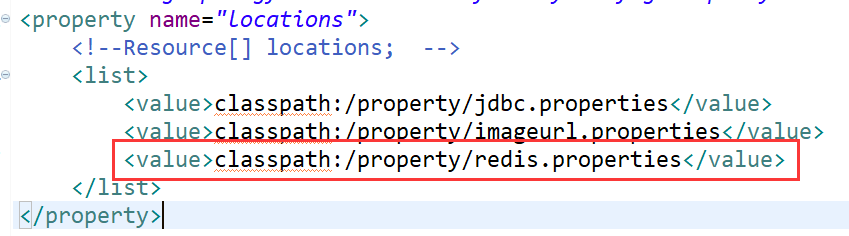
System.***out***.println(arraysUser);

}

### 编辑properties文件



### Spring加载配置文件



### Spring容器整合Redis

<!--spring容器整合redis -->

<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Controller

//实现商品分类目录展现

@RequestMapping("/list")

@ResponseBody

**public** List<EasyUITree> findItemCatList

(@RequestParam(value="id",defaultValue="0") Long parentId){

//1.查询一级商品分类信息

//Long parentId = 0L;

**return** itemCatService.findItemCatCache(parentId);

}

### 编辑Service

/\*\*

\* 1.查询缓存

\* 2.判断是否有数据

\* 3.如果没有数据查询数据库,之后将返回的结果转化为JSON串,在保存到redis

\* 4.如果查询的结果有数据,需要将json串转化为Java对象之后返回.

\*/

@Override

**public** List<EasyUITree> findItemCatCache(Long parentId) {

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

String result = jedis.get(key); //获取缓存的数据

List<EasyUITree> treeList = **null**;

**try** {

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

//证明缓存中没有数据

treeList = findItemCatList(parentId);

//将数据存入缓存

String easyUIJSON =

***objectMapper***.writeValueAsString(treeList);

jedis.set(key, easyUIJSON);

System.***out***.println("查询数据库");

}**else**{

EasyUITree[] trees = ***objectMapper***.readValue(result,EasyUITree[].**class**);

treeList = Arrays.*asList*(trees);

System.***out***.println("查询缓存");

}

} **catch** (Exception e) {

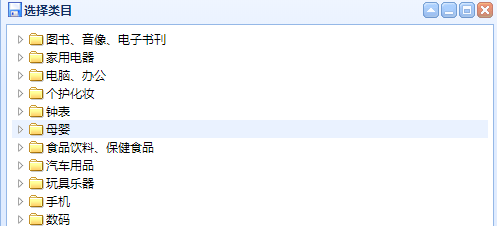
e.printStackTrace();

}

**return** treeList;

}

### 页面效果



1.不加缓存



2.使用缓存



综上可见:使用redis使程序性能提高很多.

## Redis分片介绍

### 业务需求

说明:如果单台redis出现宕机的现象.会影响整个服务.单台的redis内存受限.

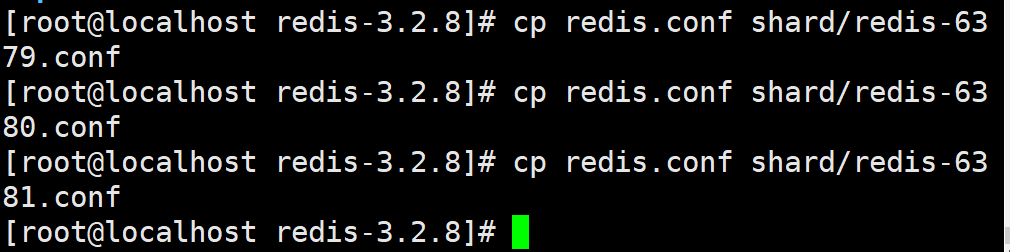
解决方法:实现redis的分片.

优点:

1. 可以实现redis的内存动态扩容
2. Redis的保存的数据,分别保存到多台redis中,如果其中有一台redis出现问题,那么其中的数据只损失1/n的数据.

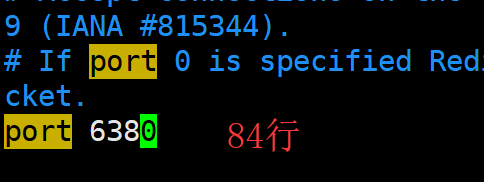
### 分片搭建

1. 新建文件shard.复制3分配置文件名称分别为redis-6379.conf,redis-6380.conf, redis-6381.conf

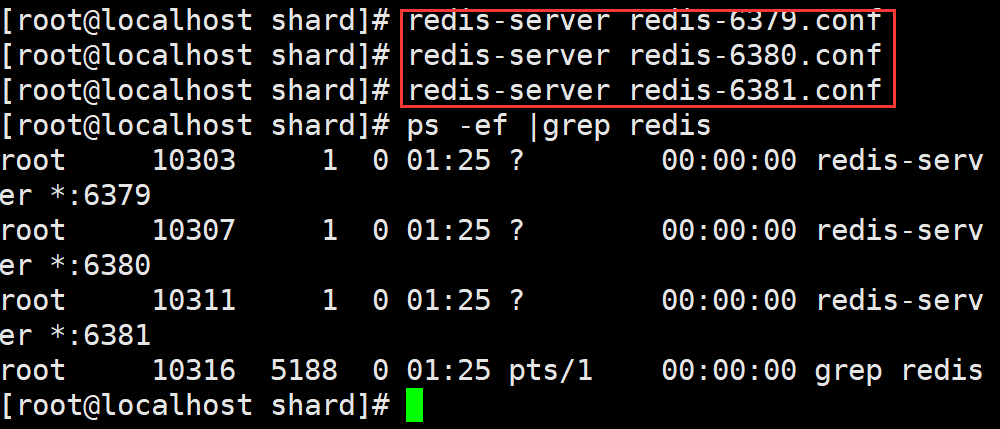


1. 修改端口

将redis的端口修改为6380/6381



1. 启动多台redis



### 分片入门案例

@Test

**public** **void** test01(){

//1.定义连接池的大小

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

poolConfig.setMaxTotal(1000);

poolConfig.setMaxIdle(100);

poolConfig.setTestOnBorrow(**true**);

//2.定义分片的List集合

List<JedisShardInfo> shards = **new** ArrayList<>();

shards.add(**new** JedisShardInfo("192.168.126.166", 6379));

shards.add(**new** JedisShardInfo("192.168.126.166", 6380));

shards.add(**new** JedisShardInfo("192.168.126.166", 6381));

//3.创建分片的对象

ShardedJedisPool jedisPool =

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

ShardedJedis shardedJedis = jedisPool.getResource();

shardedJedis.set("fenpian","分片");

System.***out***.println("获取数据:"+shardedJedis.get("fenpian"));

shardedJedis.close();

}

## Spring整合分片

### 编辑properties配置文件

redis.host=192.168.126.166

redis.port1=6379

redis.port2=6380

redis.port3=6381

redis.maxTotal=1000

redis.maxIdle=100

redis.testOnBorrow=true

### 编辑Spring配置文件

<!--1.定义poolConfigBean -->

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

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

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

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

</bean>

<!--2.定义分片的节点信息 -->

<bean id=*"shard1"* class=*"redis.clients.jedis.JedisShardInfo"*>

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

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

</bean>

<bean id=*"shard2"* class=*"redis.clients.jedis.JedisShardInfo"*>

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

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

</bean>

<bean id=*"shard3"* class=*"redis.clients.jedis.JedisShardInfo"*>

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

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

</bean>

<!--3.定义分片对象 -->

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

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

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

<list>

<ref bean=*"shard1"*/>

<ref bean=*"shard2"*/>

<ref bean=*"shard3"*/>

</list>

</constructor-arg>

</bean>

### 编辑工具类

@Service

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

//注入分片的池对象

@Autowired(required=**false**) //使用时才注入对象

**public** ShardedJedisPool shardedJedisPool;

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

ShardedJedis shardedJedis = shardedJedisPool.getResource();

shardedJedis.set(key, value);

shardedJedis.close();

}

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

ShardedJedis shardedJedis = shardedJedisPool.getResource();

String result = shardedJedis.get(key);

shardedJedis.close();

**return** result;

}

//设定超时时间的方法

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

ShardedJedis shardedJedis = shardedJedisPool.getResource();

shardedJedis.setex(key, seconds, value);

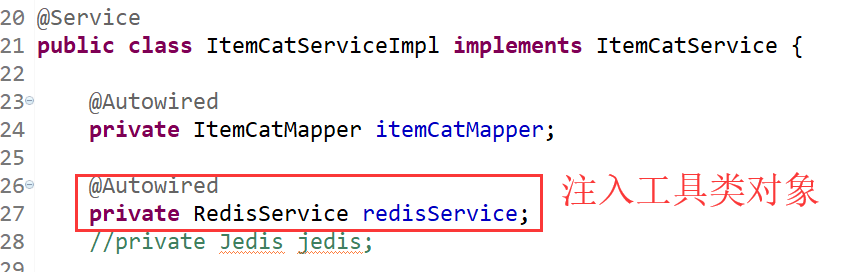
shardedJedis.close();

}

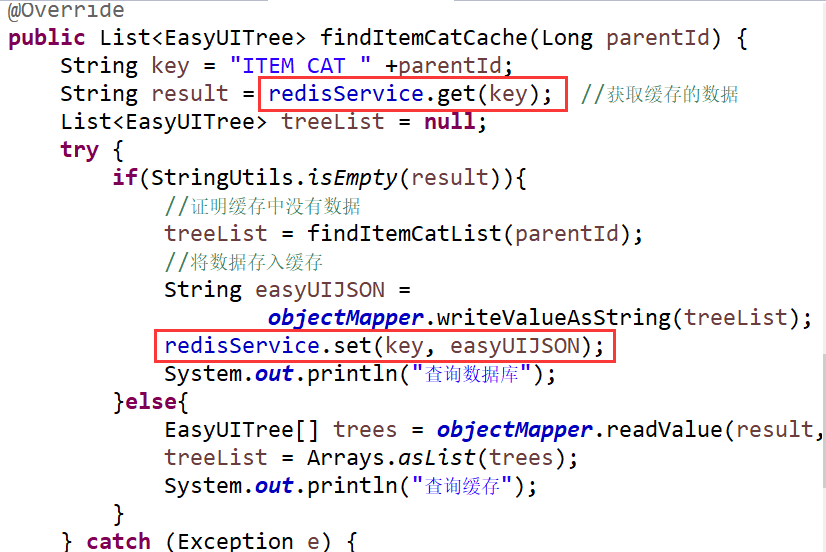
}

编辑完工具类后,将jt-common打包

### 编辑业务代码



业务代码调用.只需要修改参数名称即可



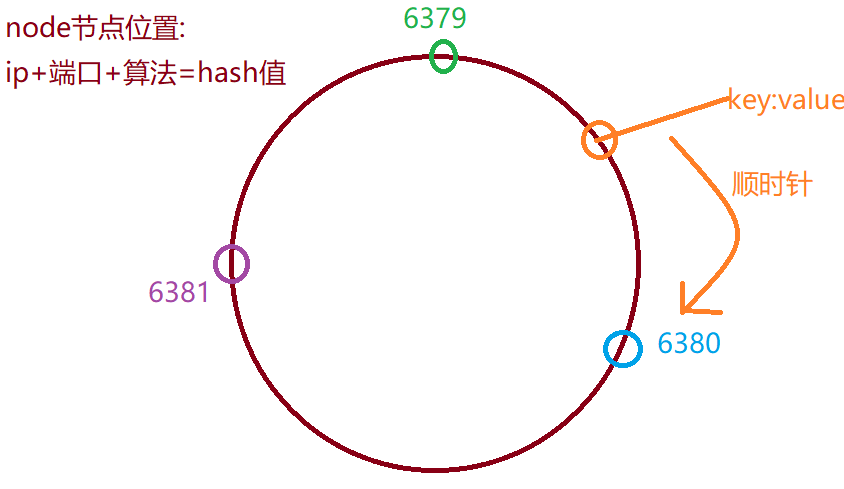
## 哈希一致性算法

说明:redis分片时采用哈希一致性算法,实现数据动态的绑定.

术语:

1.节点(node) 代表真实的redis服务器

2.key 用户保存/读取关键字



### 均衡性

问题:

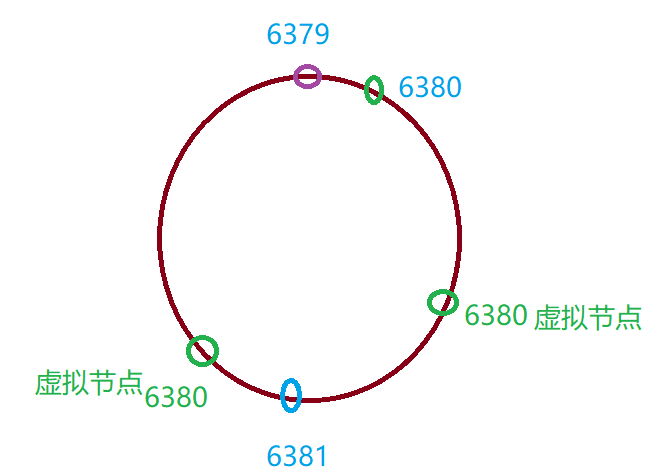
由于node节点是经过hash一致性算法计算的结果.那么可能会出现计算的节点所分配的内存不均.如果内存不均,会导致某些节点内存负载过高.

特点说明:

要求全部的数据,尽可能的均匀分配到不同的节点中,每个节点中保存的数据尽可能接近1/n

实现步骤:

如果遇到节点负载不均时,会自动的启动虚拟节点,进行数据的平衡



### 单调性

说明:如果节点的个数**增加**,原有的节点的挂载会自动的发生变化.将满足条件的数据自动的挂载到新的节点中.

原则:尽可能保证原有的数据不变

总结:节点增加,数据动态发生迁移

### 分散性

了解

### 负载

了解