# Redis集群搭建

伪分布式集群

分布式集群

# 使用redis做缓存

## 安装redis

* 版本说明

本教程使用redis4.0版本。

安装的前提条件：

需要安装gcc：yum install gcc-c++

1. 下载redis的源码包。
2. 把源码包上传到linux服务器
3. 解压源码包

tar -zxvf redis-3.0.0.tar.gz

1. Make
2. Make install

[root@bogon redis-3.0.0]# make install PREFIX=/usr/local/redis

安装后只有一个bin目录

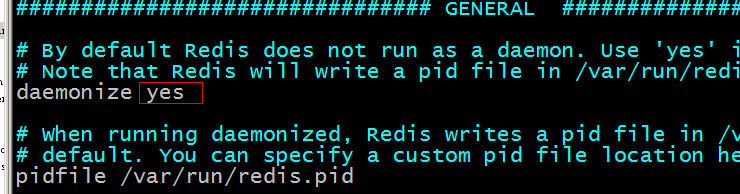
## 启动redis

1. 前端启动模式

/usr/local/redis/bin/redis-server

默认是前端启动模式，端口是6379

1. 后端启动
2. 从redis的源码目录中复制redis.conf到redis的安装目录。
3. 修改配置文件



1. [root@bogon bin]# ./redis-server redis.conf

## 添加防火墙端口

vi /etc/sysconfig/iptables

添加:

-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT

service iptables save  
service iptables restart

走一边

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

## 我们的集群结构

集群中有三个节点的集群，每个节点有一主一备。需要6台虚拟机。

搭建一个伪分布式的集群，使用6个redis实例来模拟。

## 搭建集群需要的环境

搭建集群需要使用到官方提供的ruby脚本。

需要安装ruby的环境。

-------------------------------------

 yum install ruby

继续执行上个命令，又报错“./redis-trib.rb:24:in `require': no such file to load -- rubygems (LoadError)  
        from ./redis-trib.rb:24”，需要安装rubygems

            yum install ruby  
            yum install rubygems

继续执行上个命令，又报错“/usr/lib/ruby/site\_ruby/1.8/rubygems/custom\_require.rb:31:in `gem\_original\_require': no such file to load -- redis (LoadError)  
        from /usr/lib/ruby/site\_ruby/1.8/rubygems/custom\_require.rb:31:in `require'  
        from ./redis-trib.rb:25”，需要通过gem安装redis

gem install redis

执行之后报错“ERROR:  Error installing redis:  
        redis requires Ruby version >= 2.2.2.”，需要升级ruby

8、升级ruby

1）确认rvm有没有安装

rvm -v

如果已经安装，就跳过第2步

2）安装rvm

   curl -L get.rvm.io | bash -s stable

如果没有curl命令，就执行yum install  curl进行安装

执行结果如下

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current  
                                 Dload  Upload   Total   Spent    Left  Speed  
100 24090  100 24090    0     0  21983      0  0:00:01  0:00:01 --:--:-- 21983  
Downloading https://github.com/rvm/rvm/archive/1.29.3.tar.gz  
Downloading https://github.com/rvm/rvm/releases/download/1.29.3/1.29.3.tar.gz.asc  
gpg: Signature made Mon 11 Sep 2017 04:59:21 AM CST using RSA key ID BF04FF17  
gpg: Can't check signature: No public key  
Warning, RVM 1.26.0 introduces signed releases and automated check of signatures when GPG software found. Assuming you trust Michal Papis import the mpapis public key (downloading the signatures).  
GPG signature verification failed for '/usr/local/rvm/archives/rvm-1.29.3.tgz' - 'https://github.com/rvm/rvm/releases/download/1.29.3/1.29.3.tar.gz.asc'! Try to install GPG v2 and then fetch the public key:  
    gpg2 --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3  
or if it fails:  
    command curl -sSL https://rvm.io/mpapis.asc | gpg2 --import -  
the key can be compared with:  
    https://rvm.io/mpapis.asc  
    https://keybase.io/mpapis  
NOTE: GPG version 2.1.17 have a bug which cause failures during fetching keys from remote server. Please downgrade or upgrade to newer version (if available) or use the second method described above.

按照提示，获取一下public key

 gpg2 --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

再次执行安装curl -L get.rvm.io | bash -s stable，就可以了。

3）source /usr/local/rvm/scripts/rvm

4）查看可以安装的ruby版本

rvm list known

显示如下

# MRI Rubies  
[ruby-]1.8.6[-p420]  
[ruby-]1.8.7[-head] # security released on head  
[ruby-]1.9.1[-p431]  
[ruby-]1.9.2[-p330]  
[ruby-]1.9.3[-p551]  
[ruby-]2.0.0[-p648]  
[ruby-]2.1[.10]  
[ruby-]2.2[.7]  
[ruby-]2.3[.4]  
[ruby-]2.4[.1]

5）安装2.4.1版本

rvm install 2.4.1

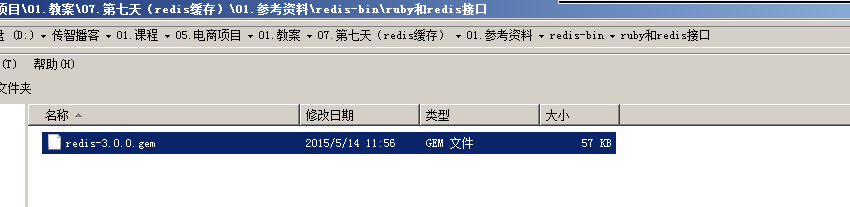
6)确认ruby版本

ruby -v

9、再次安装redis

gem install redis

(这里安装了,下面的脚本就不用安装了………..)

脚本需要的ruby包：  


需要上传到linux服务。

安装ruby的包：

gem install redis-3.0.0.gem

[root@bogon ~]# gem install redis-3.0.0.gem

Successfully installed redis-3.0.0

1 gem installed

Installing ri documentation for redis-3.0.0...

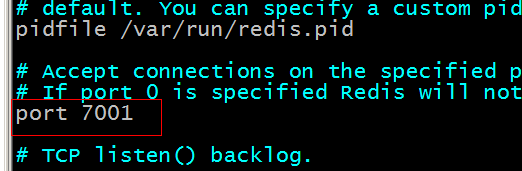
Installing RDoc documentation for redis-3.0.0...

## 集群的搭建

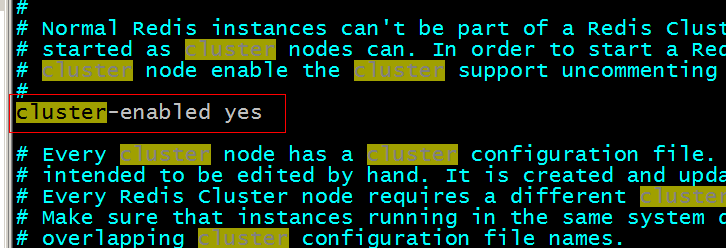
第一步：创建6个redis实例，端口号从7001~7006

第二步：修改redis的配置文件

1、修改端口号



2、打开cluster-enable前面的注释。



第三步：把创建集群的ruby脚本复制到redis-cluster目录下。

第四步：启动6个redis实例

|  |
| --- |
| [root@localhost redis-cluster]# cat startall.sh  cd redis01  ./redis-server redis.conf  cd ..  cd redis02  ./redis-server redis.conf  cd ..  cd redis03  ./redis-server redis.conf  cd ..  cd redis04  ./redis-server redis.conf  cd ..  cd redis05  ./redis-server redis.conf  cd ..  cd redis06  ./redis-server redis.conf |

|  |
| --- |
| [root@localhost redis-cluster]# cat shutdownall.sh  redis01/redis-cli -p 7001 shutdown  redis02/redis-cli -p 7002 shutdown  redis03/redis-cli -p 7003 shutdown  redis04/redis-cli -p 7004 shutdown  redis05/redis-cli -p 7005 shutdown  redis06/redis-cli -p 7006 shutdown |

第五步：创建集群。

|  |
| --- |
| ./redis-trib.rb create --replicas 1 192.168.106.126:7001 192.168.106.126:7002 192.168.106.126:7003 192.168.106.126:7004 192.168.106.126:7005 192.168.106.126:7006 |

[root@bogon redis-cluster]# ./redis-trib.rb create --replicas 1 192.168.25.153:7001 192.168.25.153:7002 192.168.25.153:7003 192.168.25.153:7004 192.168.25.153:7005 192.168.25.153:7006

>>> Creating cluster

Connecting to node 192.168.25.153:7001: OK

Connecting to node 192.168.25.153:7002: OK

Connecting to node 192.168.25.153:7003: OK

Connecting to node 192.168.25.153:7004: OK

Connecting to node 192.168.25.153:7005: OK

Connecting to node 192.168.25.153:7006: OK

>>> Performing hash slots allocation on 6 nodes...

Using 3 masters:

192.168.25.153:7001

192.168.25.153:7002

192.168.25.153:7003

Adding replica 192.168.25.153:7004 to 192.168.25.153:7001

Adding replica 192.168.25.153:7005 to 192.168.25.153:7002

Adding replica 192.168.25.153:7006 to 192.168.25.153:7003

M: 5a8523db7e12ca600dc82901ced06741b3010076 192.168.25.153:7001

slots:0-5460 (5461 slots) master

M: bf6f0929044db485dea9b565bb51e0c917d20a53 192.168.25.153:7002

slots:5461-10922 (5462 slots) master

M: c5e334dc4a53f655cb98fa3c3bdef8a808a693ca 192.168.25.153:7003

slots:10923-16383 (5461 slots) master

S: 2a61b87b49e5b1c84092918fa2467dd70fec115f 192.168.25.153:7004

replicates 5a8523db7e12ca600dc82901ced06741b3010076

S: 14848b8c813766387cfd77229bd2d1ffd6ac8d65 192.168.25.153:7005

replicates bf6f0929044db485dea9b565bb51e0c917d20a53

S: 3192cbe437fe67bbde9062f59d5a77dabcd0d632 192.168.25.153:7006

replicates c5e334dc4a53f655cb98fa3c3bdef8a808a693ca

Can I set the above configuration? (type 'yes' to accept): **yes**

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join.....

>>> Performing Cluster Check (using node 192.168.25.153:7001)

M: 5a8523db7e12ca600dc82901ced06741b3010076 192.168.25.153:7001

slots:0-5460 (5461 slots) master

M: bf6f0929044db485dea9b565bb51e0c917d20a53 192.168.25.153:7002

slots:5461-10922 (5462 slots) master

M: c5e334dc4a53f655cb98fa3c3bdef8a808a693ca 192.168.25.153:7003

slots:10923-16383 (5461 slots) master

M: 2a61b87b49e5b1c84092918fa2467dd70fec115f 192.168.25.153:7004

slots: (0 slots) master

replicates 5a8523db7e12ca600dc82901ced06741b3010076

M: 14848b8c813766387cfd77229bd2d1ffd6ac8d65 192.168.25.153:7005

slots: (0 slots) master

replicates bf6f0929044db485dea9b565bb51e0c917d20a53

M: 3192cbe437fe67bbde9062f59d5a77dabcd0d632 192.168.25.153:7006

slots: (0 slots) master

replicates c5e334dc4a53f655cb98fa3c3bdef8a808a693ca

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

[root@bogon redis-cluster]#

## 测试集群

[root@bogon redis-cluster]# redis01/redis-cli -h 192.168.106.126 -p 7002 **-c**

[root@bogon redis-cluster]# redis01/redis-cli -h 192.168.25.153 -p 7002

192.168.25.153:7002> set a 100

(error) MOVED 15495 192.168.25.153:7003

192.168.25.153:7002>

[root@bogon redis-cluster]# redis01/redis-cli -h 192.168.25.153 -p 7002 -c

192.168.25.153:7002> set a 100

-> Redirected to slot [15495] located at 192.168.25.153:7003

OK

192.168.25.153:7003>

## 关闭redis

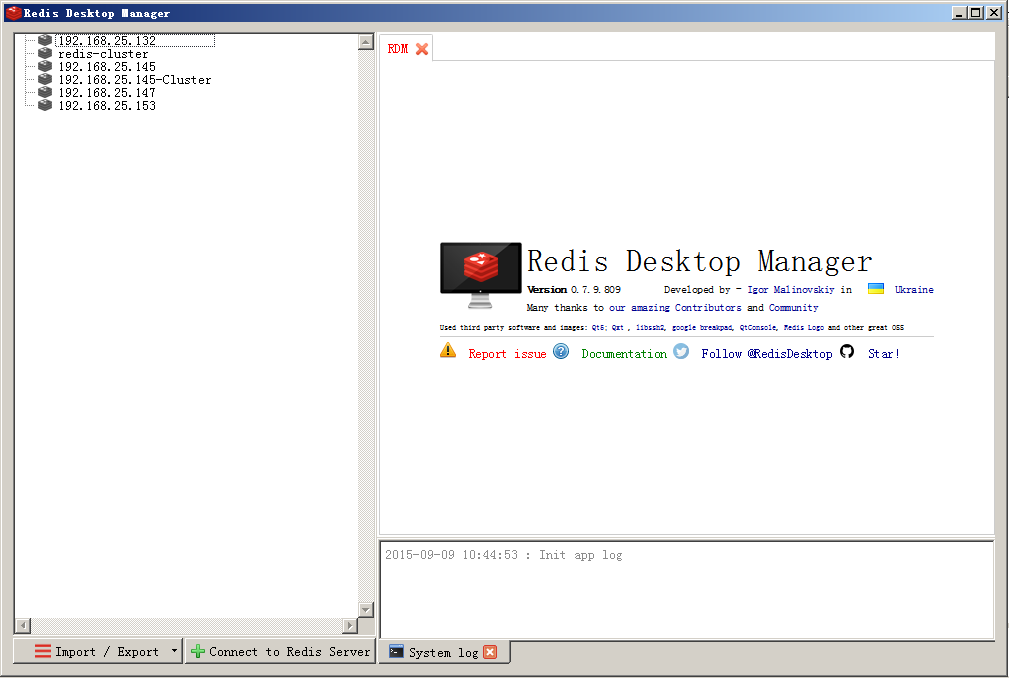
redis01/redis-cli -p 7001 shutdown

## Redis客户端

### Redis-cli

自带客户端。使用最多的。

### 图形化界面客户端



只支持单机版，不支持集群。

### Jedis客户端

#### 单机版

需要把jedis的jar包添加到工程中，如果是maven需要添加jar包的坐标。

<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->

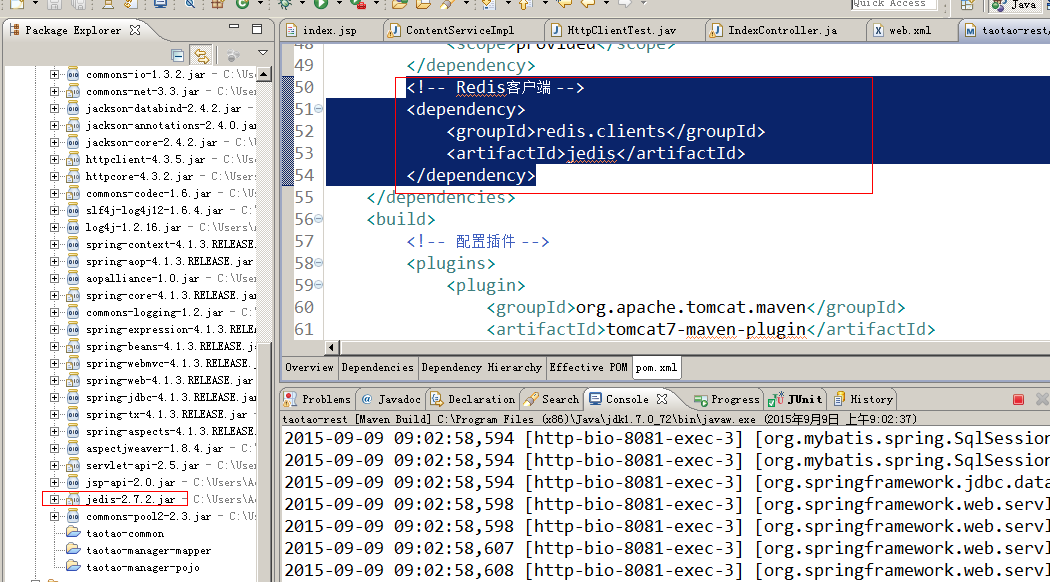
<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

</dependency>



|  |
| --- |
| **public** **class** JedisTest {  @Test  **public** **void** testJedisSingle() {  //创建一个jedis的对象。  Jedis jedis = **new** Jedis("192.168.25.153", 6379);  //调用jedis对象的方法，方法名称和redis的命令一致。  jedis.set("key1", "jedis test");  String string = jedis.get("key1");  System.***out***.println(string);  //关闭jedis。  jedis.close();  }    /\*\*  \* 使用连接池  \*/  @Test  **public** **void** testJedisPool() {  //创建jedis连接池  JedisPool pool = **new** JedisPool("192.168.25.153", 6379);  //从连接池中获得Jedis对象  Jedis jedis = pool.getResource();  String string = jedis.get("key1");  System.***out***.println(string);  //关闭jedis对象  jedis.close();  pool.close();  }  } |

#### 集群版

|  |
| --- |
| @Test  **public** **void** testJedisCluster() {  HashSet<HostAndPort> nodes = **new** HashSet<>();  nodes.add(**new** HostAndPort("192.168.25.153", 7001));  nodes.add(**new** HostAndPort("192.168.25.153", 7002));  nodes.add(**new** HostAndPort("192.168.25.153", 7003));  nodes.add(**new** HostAndPort("192.168.25.153", 7004));  nodes.add(**new** HostAndPort("192.168.25.153", 7005));  nodes.add(**new** HostAndPort("192.168.25.153", 7006));    JedisCluster cluster = **new** JedisCluster(nodes);    cluster.set("key1", "1000");  String string = cluster.get("key1");  System.***out***.println(string);    cluster.close();  } |

# 业务逻辑中添加缓存

需要在taotao-rest工程中添加缓存。

## jedis整合spring

### 单机版整合

#### 配置

|  |
| --- |
| <!-- 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"* />  <!-- 释放连接的扫描间隔（毫秒） -->  <property name=*"timeBetweenEvictionRunsMillis"* value=*"30000"* />  <!-- 连接最小空闲时间 -->  <property name=*"minEvictableIdleTimeMillis"* value=*"1800000"* />  <!-- 连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放 -->  <property name=*"softMinEvictableIdleTimeMillis"* value=*"10000"* />  <!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->  <property name=*"maxWaitMillis"* value=*"1500"* />  <!-- 在获取连接的时候检查有效性, 默认false -->  <property name=*"testOnBorrow"* value=*"true"* />  <!-- 在空闲时检查有效性, 默认false -->  <property name=*"testWhileIdle"* value=*"true"* />  <!-- 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"* />  </bean>  <!-- jedis客户端单机版 -->  <bean id=*"redisClient"* class=*"redis.clients.jedis.JedisPool"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"6379"*></constructor-arg>  <constructor-arg name=*"poolConfig"* ref=*"jedisPoolConfig"*></constructor-arg>  </bean> |

#### 测试

|  |
| --- |
| /\*\*  \* 单机版测试  \* <p>Title: testSpringJedisSingle</p>  \* <p>Description: </p>  \*/  @Test  **public** **void** testSpringJedisSingle() {  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  JedisPool pool = (JedisPool) applicationContext.getBean("redisClient");  Jedis jedis = pool.getResource();  String string = jedis.get("key1");  System.***out***.println(string);  jedis.close();  pool.close();  } |

### 集群版整合

#### 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=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7001"*></constructor-arg>  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7002"*></constructor-arg>  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7003"*></constructor-arg>  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7004"*></constructor-arg>  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7005"*></constructor-arg>  </bean>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.25.153"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7006"*></constructor-arg>  </bean>  </set>  </constructor-arg>  <constructor-arg name=*"poolConfig"* ref=*"jedisPoolConfig"*></constructor-arg>  </bean> |

#### 测试

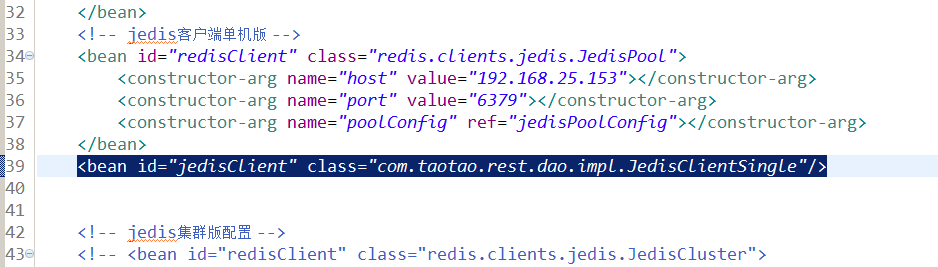
|  |
| --- |
| @Test  **public** **void** testSpringJedisCluster() {  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  JedisCluster jedisCluster = (JedisCluster) applicationContext.getBean("redisClient");  String string = jedisCluster.get("key1");  System.***out***.println(string);  jedisCluster.close();  } |

## 添加jedis dao

### 单机版

|  |
| --- |
| **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;  }  } |

Spring配置文件



### 集群版

|  |
| --- |
| **public** **class** JedisClientCluster **implements** JedisClient {  @Autowired  **private** JedisCluster jedisCluster;    @Override  **public** String get(String key) {  **return** jedisCluster.get(key);  }  @Override  **public** String set(String key, String value) {  **return** jedisCluster.set(key, value);  }  @Override  **public** String hget(String hkey, String key) {  **return** jedisCluster.hget(hkey, key);  }  @Override  **public** **long** hset(String hkey, String key, String value) {  **return** jedisCluster.hset(hkey, key, value);  }  @Override  **public** **long** incr(String key) {  **return** jedisCluster.incr(key);  }  @Override  **public** **long** expire(String key, **int** second) {  **return** jedisCluster.expire(key, second);  }  @Override  **public** **long** ttl(String key) {  **return** jedisCluster.ttl(key);  }  } |

## 把缓存添加到业务逻辑

注意：缓存的添加不能影响正常的业务逻辑。

|  |
| --- |
| @Override  **public** List<TbContent> getContentList(**long** contentCid) {  //从缓存中取内容  **try** {  String result = jedisClient.hget(INDEX\_CONTENT\_REDIS\_KEY, contentCid + "");  **if** (!StringUtils.*isBlank*(result)) {  //把字符串转换成list  List<TbContent> resultList = JsonUtils.*jsonToList*(result, TbContent.**class**);  **return** resultList;  }  } **catch** (Exception e) {  e.printStackTrace();  }    //根据内容分类id查询内容列表  TbContentExample example = **new** TbContentExample();  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(contentCid);  //执行查询  List<TbContent> list = contentMapper.selectByExample(example);    //向缓存中添加内容  **try** {  //把list转换成字符串  String cacheString = JsonUtils.*objectToJson*(list);  jedisClient.hset(INDEX\_CONTENT\_REDIS\_KEY, contentCid + "", cacheString);  } **catch** (Exception e) {  e.printStackTrace();  }    **return** list;  } |

**课后作业：商品类目展示添加缓存**

# 缓存同步

当后台管理系统，修改内容之后需要通知redis把修改的内容对应的分类id的key删除。

## 添加缓存后的系统架构

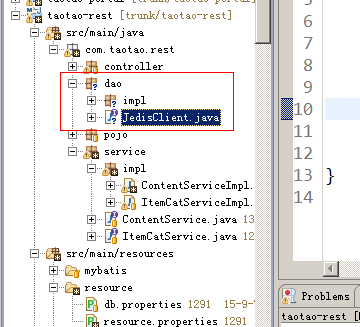


## 解决方案

在taotao-rest工程中发布一个服务。当后台管理系统修改内容后，调用此服务，同步缓存。

## Dao层

使用JedisClient接口对应的实现类。



## Service层

接收内容分类id，调用dao删除redis中对应的hash中key为分类id的项。

参数：内容分类id

返回值：TaotaoResult

|  |
| --- |
| @Service  **public** **class** RedisServiceImpl **implements** RedisService {  @Autowired  **private** JedisClient jedisClient;    @Value("${INDEX\_CONTENT\_REDIS\_KEY}")  **private** String INDEX\_CONTENT\_REDIS\_KEY;    @Override  **public** TaotaoResult syncContent(**long** contentCid) {  **try** {  jedisClient.hdel(INDEX\_CONTENT\_REDIS\_KEY, contentCid + "");  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  **return** TaotaoResult.*ok*();  }  } |

## Controller层

接收内容分类id，调用Service返回taotaoResult。

|  |
| --- |
| @Controller  @RequestMapping("/cache/sync")  **public** **class** RedisController {  @Autowired  **private** RedisService redisService;    @RequestMapping("/content/{contentCid}")  **public** TaotaoResult contentCacheSync(@PathVariable Long contentCid) {  TaotaoResult result = redisService.syncContent(contentCid);  **return** result;  }  } |

## 同步缓存服务的调用

需要在后台管理系统中添加一个服务调用的逻辑。当修改内容信息后，需要调用此服务同步缓存。

