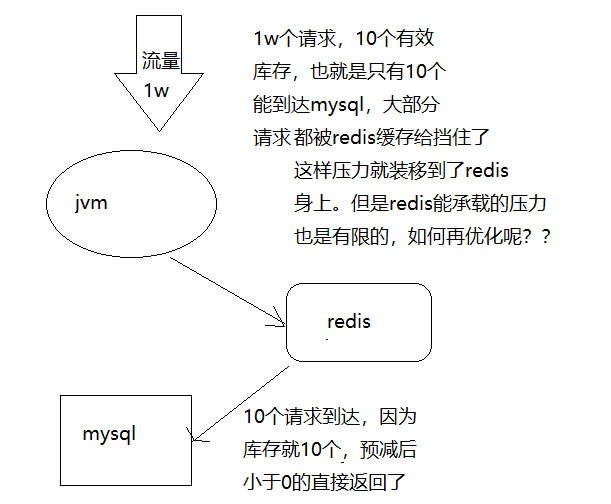
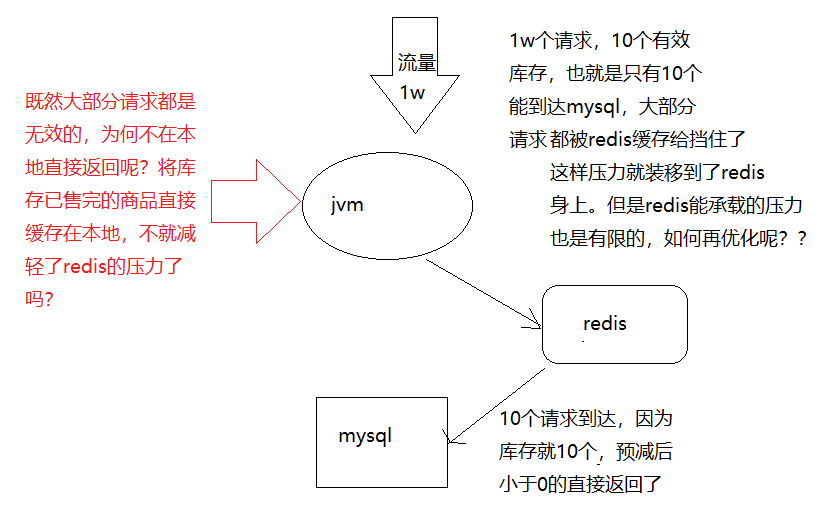
# redis问题



Redis也扛不住了，怎么办啊？？

# 售完标记



## 并发问题

为什么要使用并发安全的本地缓存？？？？

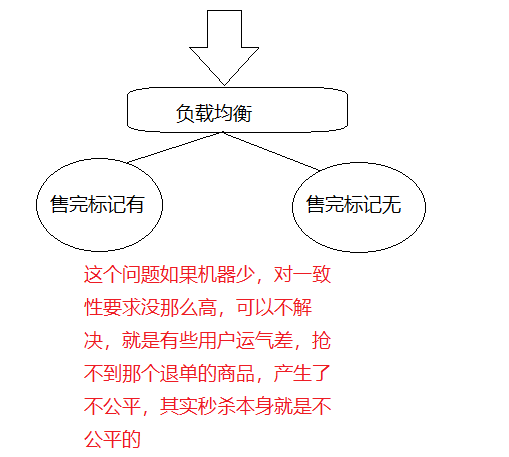
**因为该缓存对象为堆内对象，为线程共享，所以要考虑并发安全问题！！**

## 一致性问题

上面只是解决了堆内并发安全问题，那么分布式下的并发安全吗？？



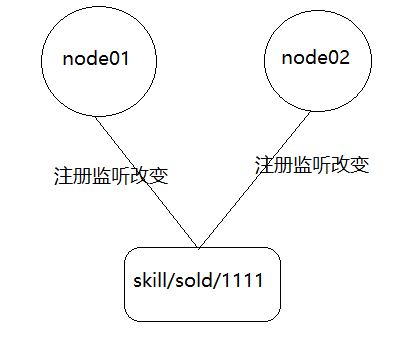
## 算bug吗



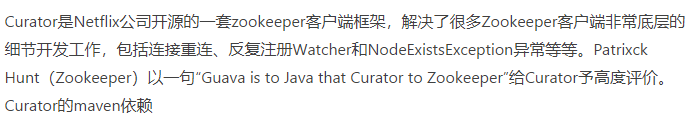
# 缓存一致性

## 思路

分布式数据一致性问题：可以使用分布式协同工具，zk来解决！！！！！



## Zk



**“永久注册”不存在的！！是框架内部做处理。**

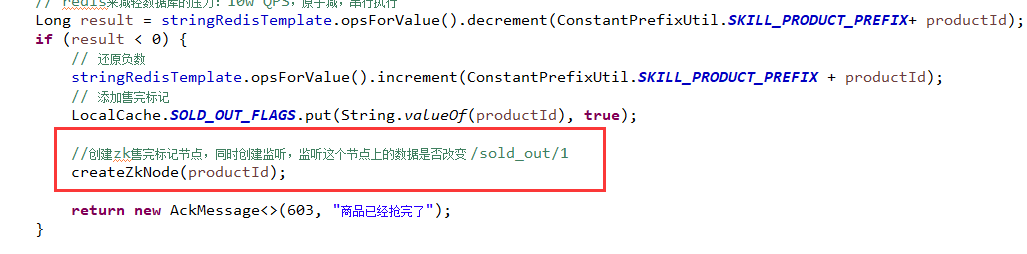
|  |
| --- |
| <!-- 对zookeeper的底层api的一些封装 -->  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-framework</artifactId>  <version>2.12.0</version>  </dependency>  <!-- 封装了一些高级特性，如：Cache事件监听、选举、分布式锁、分布式Barrier -->  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-recipes</artifactId>  <version>2.12.0</version>  </dependency> |

简单测试代码

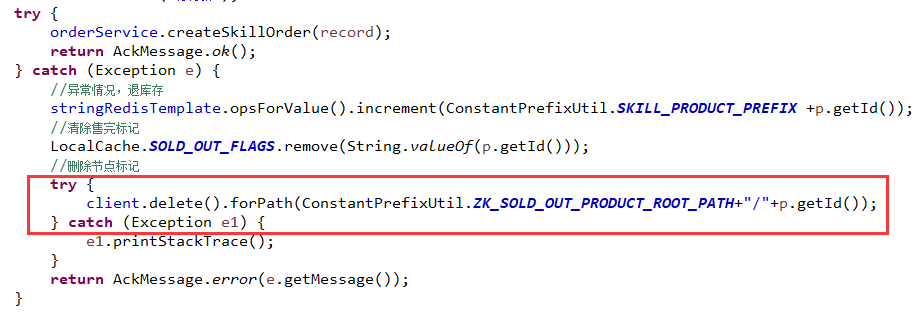
|  |
| --- |
| package com.cjl.skill.test;  import org.apache.curator.RetryPolicy;  import org.apache.curator.framework.CuratorFramework;  import org.apache.curator.framework.CuratorFrameworkFactory;  import org.apache.curator.framework.recipes.cache.NodeCache;  import org.apache.curator.framework.recipes.cache.NodeCacheListener;  import org.apache.curator.retry.ExponentialBackoffRetry;  import org.apache.zookeeper.CreateMode;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  @RunWith(SpringJUnit4ClassRunner.class)  @SpringBootTest  public class ZkTest {  @Test  public void testZkCreate() throws Exception {  RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);  CuratorFramework client = CuratorFrameworkFactory.newClient("127.0.0.1:2181",  5000, 5000, retryPolicy);  client.start();  client.create().creatingParentContainersIfNeeded() // 递归创建所需父节点  .withMode(CreateMode.PERSISTENT) // 创建类型为持久节点  .forPath("/testzk/nodeA", "init".getBytes()); // 目录及内容  }    @Test  public void testZkWatcher() throws Exception {  RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);  CuratorFramework client = CuratorFrameworkFactory.newClient("127.0.0.1:2181",  5000, 5000, retryPolicy);  client.start();  /\*Curator之nodeCache一次注册，N次监听\*/  //为节点添加watcher  //监听数据节点的变更，会触发事件  final NodeCache nodeCache = new NodeCache(client,"/testzk/nodeA");  //buildInitial: 初始化的时候获取node的值并且缓存  nodeCache.start(true);  if(nodeCache.getCurrentData() != null){  System.out.println("节点的初始化数据为："+new String(nodeCache.getCurrentData().getData()));  }else{  System.out.println("节点初始化数据为空。。。");  }  nodeCache.getListenable().addListener(new NodeCacheListener() {  public void nodeChanged() throws Exception {  //获取当前数据  String data = new String(nodeCache.getCurrentData().getData());  System.out.println("节点路径为："+nodeCache.getCurrentData().getPath()+" 数据: "+data);  }  });  Thread.sleep(60000);  }  } |

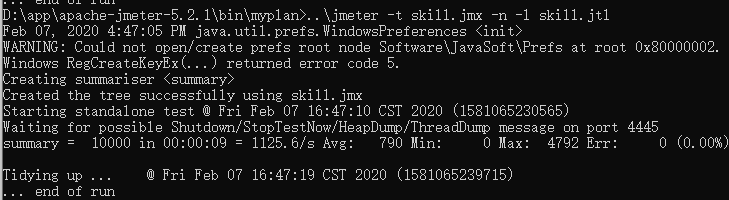
## 缓存同步

|  |
| --- |
| package com.cjl.skill.config;  import org.apache.curator.RetryPolicy;  import org.apache.curator.framework.CuratorFramework;  import org.apache.curator.framework.CuratorFrameworkFactory;  import org.apache.curator.framework.recipes.cache.ChildData;  import org.apache.curator.framework.recipes.cache.NodeCache;  import org.apache.curator.framework.recipes.cache.NodeCacheListener;  import org.apache.curator.framework.recipes.cache.TreeCache;  import org.apache.curator.framework.recipes.cache.TreeCacheEvent;  import org.apache.curator.framework.recipes.cache.TreeCacheListener;  import org.apache.curator.retry.ExponentialBackoffRetry;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import com.cjl.skill.cache.LocalCache;  import com.cjl.skill.util.ConstantPrefixUtil;  @Configuration  public class ZookeeperConfig {    @Bean  public CuratorFramework zookeeperClient() {  RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);  CuratorFramework client = CuratorFrameworkFactory.newClient("127.0.0.1:2181",  5000, 5000, retryPolicy);  client.start();    /\* Curator之nodeCache一次注册，N次监听 \*/  // 为节点添加watcher  // 监听数据节点的变更，会触发事件  TreeCache treeCache = new TreeCache(client, ConstantPrefixUtil.ZK\_SOLD\_OUT\_PRODUCT\_ROOT\_PATH);  // buildInitial: 初始化的时候获取node的值并且缓存  try {  treeCache.start();  } catch (Exception e) {  e.printStackTrace();  }    treeCache.getListenable().addListener(new TreeCacheListener() {  @Override  public void childEvent(CuratorFramework client, TreeCacheEvent event) throws Exception {  ChildData eventData = event.getData();  String full = eventData.getPath();  String id = full.substring(full.lastIndexOf("/")+1);  switch (event.getType()) {  case NODE\_ADDED:  LocalCache.SOLD\_OUT\_FLAGS.put(id, true);  System.out.println("node add : "+"full path: "+full+" id : "+id);  break;  case NODE\_REMOVED:  LocalCache.SOLD\_OUT\_FLAGS.remove(id);  System.out.println("node removed : "+"full path: "+full+" id : "+id);  break;  default:  break;  }  }  });  return client;  }  } |



|  |
| --- |
| // 创建zknode，用于同步jvm缓存  **private** **void** createZkNode(**int** productId) {  **try** {  //没有就创建  **if** (client.checkExists().forPath(ConstantPrefixUtil.***ZK\_SOLD\_OUT\_PRODUCT\_ROOT\_PATH*** +"/"+ productId) == **null**) {  client.create().creatingParentContainersIfNeeded().withMode(CreateMode.***PERSISTENT***)  .withACL(ZooDefs.Ids.***OPEN\_ACL\_UNSAFE***)  .forPath(ConstantPrefixUtil.***ZK\_SOLD\_OUT\_PRODUCT\_ROOT\_PATH*** +"/"+ productId);  } **else** {  //有人退单了，后面的线程又抢掉了这个库存，之后；注意这个细节。  client.setData().forPath(ConstantPrefixUtil.***ZK\_SOLD\_OUT\_PRODUCT\_ROOT\_PATH*** +"/"+ productId);  }  } **catch** (Exception e) {  e.printStackTrace();  }  } |





## 问题思考

