# Echcache

## 简介

Ehcahce有三层。

1. 最上层chcheManager，他是操作ehcache的入口。每个cacheManager都可以管理多个cache（即配置文件中的cache），存放多组缓存数据。
2. 第二层cache，cache是操作element的入口，是存放element的地方，一个cache存放一组缓存数据。
3. 底层element，element是我们存放缓存内容的地方，每个element存放一个缓存数据。

注：Ehcache被缓存的对象需要实现Serializable接口

## 引入jar包

ehcache-2.1.0-distribution.tar.gz和ehcache-web-2.0.2-distribution.tar.gz解压的jar包放入WEB-INF/lib下

## Api

1. 获取cacheManger的方式
   1. (Create a singleton CacheManager using defaults, then list caches.  
      CacheManager.create();  
      String[] cacheNames = CacheManager.getInstance().getCacheNames();
   2. Create a CacheManager instance using defaults, then list caches.  
      CacheManager manager = new CacheManager();  
      String[] cacheNames = manager.getCacheNames();
   3. (3)Create two CacheManagers, each with a different configuration, and list the caches in each.  
      CacheManager manager1 = new CacheManager("src/config/ehcache1.xml");  
      CacheManager manager2 = new CacheManager("src/config/ehcache2.xml");  
      String[] cacheNamesForManager1 = manager1.getCacheNames();  
      String[] cacheNamesForManager2 = manager2.getCacheNames();
2. CacheManager 的构造函数如下：
   1. 无参  
      CacheManager manager = new CacheManager();
   2. 通过配置文件  
      CacheManager manager = new CacheManager("src/config/ehcache.xml");
   3. 通过资源  
      URL url = getClass().getResource("/anotherconfigurationname.xml");  
      CacheManager manager = new CacheManager(url);
   4. 通过输入流  
      InputStream fis = new FileInputStream(new File("src/config/ehcache.xml").getAbsolutePath());  
      try {  
       CacheManager manager = new CacheManager(fis);  
      } finally {  
       fis.close();  
      }
3. 增加或删除cache
   1. 增加cache
      1. 增加cache并加入到CacheManager  
         CacheManager singletonManager = CacheManager.create();  
         singletonManager.addCache("testCache");  
         Cache test = singletonManager.getCache("testCache");
      2. 新增一个 Cache ，然后加到 CacheManager 中， Cache 在加入 CacheManager 之前是不能使用的  
         CacheManager singletonManager = CacheManager.create();  
         Cache memoryOnlyCache = new Cache("testCache", 5000, false, false, 5, 2);  
         manager.addCache(memoryOnlyCache);  
         Cache test = singletonManager.getCache("testCache");
   2. 删除cache  
      CacheManager singletonManager = CacheManager.create();  
      singletonManager.removeCache("sampleCache1");
4. **关闭 CacheManager**  
   CacheManager 在使用之后应该关闭，虽然有自己的 shutdown hook ，建议在程序中手动关闭。  
   CacheManager.getInstance().shutdown();

## 配置文件

建立ehcache.xml放到src目录下。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <ehcache  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation=" ehcache.xsd"  updateCheck="true"  monitoring="autodetect"  dynamicConfig="true">  <!--timeToIdleSeconds 当缓存闲置n秒后销毁 -->  <!--timeToLiveSeconds 当缓存存活n秒后销毁 -->  <!--  缓存配置  name:缓存名称。  maxElementsInMemory：缓存最大个数。  eternal:对象是否永久有效，一但设置了，timeout将不起作用。  timeToIdleSeconds：设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。  timeToLiveSeconds：设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。  overflowToDisk：当内存中对象数量达到maxElementsInMemory时，Ehcache将会对象写到磁盘中。  diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。  maxElementsOnDisk：硬盘最大缓存个数。  diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.  diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。  memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。  clearOnFlush：内存数量最大时是否清除。  -->  <--默认的cache 注：在ehcache配置文件中必须有defaultCache，每个<cache>标签定义一个新的cache，属性的含义基本上可以从名字上得到。每个-->  <defaultCache  maxElementsInMemory="2"  eternal="false"  timeToIdleSeconds="1"  timeToLiveSeconds="1"  overflowToDisk="false"  memoryStoreEvictionPolicy="LRU"/>  <--自定义的的cache -->  <cache name="sampleCache1"  maxElementsInMemory="5"  eternal="false"  overflowToDisk="false"  timeToIdleSeconds="1"  timeToLiveSeconds="1"  memoryStoreEvictionPolicy="LRU"/>  </ehcache> |
|  |

## 实例程序

|  |
| --- |
| **package** ehcache.test01;  **import** java.util.List;  **import** net.sf.ehcache.CacheManager;  **import** net.sf.ehcache.Cache;  **import** net.sf.ehcache.Element;  **public** **class** test01 {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) **throws** Exception{  // **TODO** Auto-generated method stub  //使用ehcache.xml配置文件创建一个缓存管理器  CacheManager manager = **new** CacheManager("ehcache.xml");  //获得ehcache.xml配置文件中sampleCache1缓存的句柄  Cache cache = manager.getCache("sampleCache1");    **for**(**int** i = 0; i< 10 ; i++){  Element e = **new** Element("key"+ i ,"value" + i );  System.*out*.println(e.toString());  cache.put(e);  }  //更新一条记录  cache.put(**new** Element("key7","NewValue7"));    //根据key取得对应element的序列化value值  System.*out*.println("Get Serializable value : " + cache.get("key7").getValue().toString());    //根据key取得对应element的非序列化value值  System.*out*.println("Get Non Serializable value : " + cache.get("key7").getObjectValue().toString());  //SampleCache1的配置是支持磁盘持久化的。如果想要保证element即时的被输出到磁盘，可以调用cache.flush();  //cache.flush();    //获得当前cache中的element数量  System.*out*.println("cache size =　" + cache.getSize());    //获得当前MemoryStore中的element数量  System.*out*.println("cache memorystore size =　" + cache.getMemoryStoreSize());    //获得当前DiskStore中element数量  System.*out*.println("cache Diskstore size =　" + cache.getDiskStoreSize());        List<String> keys = cache.getKeys();  //System.out.println(cache.getKeys().size());  **for**(String key : keys){  System.*out*.println(key + "," + cache.get(key));  }    }  } |

## 实际运用

### 页面缓存

一个网站的首页估计是被访问次数最多的，我们可以考虑给首页做一个页面缓存；

缓存策略：应该是某个固定时间之内不变的，比如说2分钟更新一次，以应用结构page-filter-action-service-dao-db为例。

位置：页面缓存做到尽量靠近客户的地方，就是在page和filter之间，这样的优点就是第一个用户请求后，页面被缓存，第二个用户在请求，走到filter这个请求就结束了，需要在走到action-service-dao-db，好处当然是服务器压力大大降低和客户端页面响应速度加快。

首页页面缓存存活时间定为2分钟，也就是参数timeToLiveSeconds（缓存的存活时间）应该设置为120，同时timeToIdleSeconds（多长时间不访问缓存，就清除该缓存）最好也设为2分钟或者小于2分钟。

接着我们来看一下SimplePageCachingFilter 的配置，

|  |
| --- |
| <filter>  <filter-name>indexCacheFilterfilter-name>  <filter-class>  net.sf.ehcache.constructs.web.filter.SimplePageCachingFilter  <filter-class>  <filter>  <filter-mapping>  <filter-name>indexCacheFilterfilter-name>  <url-pattern>\*index.action<url-pattern>  <filter-mapping> |

将上述代码加入到web.xml，那么当打开首页时，你会发现2分钟才会有一堆sql语句出现在控制台，也可以调整为5分钟，总之一切尽在掌控之中。

当然，如果你想缓存首页的部分内容时，你需要使用SimplePageFragmentCachingFilter这个filter，我看一下：

|  |
| --- |
| <filter>  <filter-name>indexCacheFilterfilter-name>  <filter-class>  net.sf.ehcache.constructs.web.filter.SimplePageFragmentCachingFilter  <filter-class>  filter>  <filter-mapping>  <filter-name>indexCacheFilterfilter-name>  <url-pattern>\*/index\_right.jsp<url-pattern>  <filter-mapping> |

如此我们将jsp页面通过jsp：include到其他页面，这样就做到了页面局部缓存的效果，这一点貌似没有oscache的tag好用。

此外cachefilter中还有一个特性，就是gzip，也就是缓存中的元素是被压缩过的，如果客户端浏览器支持压缩的话，filter会直接返回压缩过的流，这样节省了带宽，把解压的工作交给了客户端浏览即可，当然如果客户端不支持gzip，那么filter会把缓存的元素拿出来解压后在返回给客户端浏览器（大多数爬虫是不支持gzip的，所以filter也会解压后在返回流）。

总之，Ehcache是一个非常轻量级的缓存实现，而且从1.2之后支持了集群，而且是hibernate默认的缓存provider，本文主要介绍Ehcahe对页面缓存的支持，但是它的功能远不止如此，要用好缓存，对J2ee中缓存的原理、适用范围、适用场景等等都需要比较深刻的理解，这样才能用好用对缓

### Spring整合EhCache

#### ehcache.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">      <!--<diskStore>：当内存缓存中对象数量超过maxElementsInMemory时,将缓存对象写到磁盘缓存中(需对象实现序列化接口)。  <diskStore path="">：用来配置磁盘缓存使用的物理路径,Ehcache磁盘缓存使用的文件后缀名是\*.data和\*.index。  name：缓存名称,cache的唯一标识(ehcache会把这个cache放到HashMap里)。  maxElementsOnDisk：磁盘缓存中最多可以存放的元素数量,0表示无穷大。  maxElementsInMemory：内存缓存中最多可以存放的元素数量,若放入Cache中的元素超过这个数值,则有以下两种情况。  1)若overflowToDisk=true,则会将Cache中多出的元素放入磁盘文件中。  2)若overflowToDisk=false,则根据memoryStoreEvictionPolicy策略替换Cache中原有的元素。  Eternal：缓存中对象是否永久有效,即是否永驻内存,true时将忽略timeToIdleSeconds和timeToLiveSeconds。  timeToIdleSeconds：缓存数据在失效前的允许闲置时间(单位:秒),仅当eternal=false时使用,默认值是0表示可闲置时间无穷大,此为可选属性即访问这个cache中元素的最大间隔时间,若超过这个时间没有访问此Cache中的某个元素,那么此元素将被从Cache中清除。  timeToLiveSeconds：缓存数据在失效前的允许存活时间(单位:秒),仅当eternal=false时使用,默认值是0表示可存活时间无穷大，即Cache中的某元素从创建到清楚的生存时间,也就是说从创建开始计时,当超过这个时间时,此元素将从Cache中清除。  overflowToDisk：内存不足时,是否启用磁盘缓存(即内存中对象数量达到maxElementsInMemory时,Ehcache会将对象写到磁盘中)，会根据标签中path值查找对应的属性值,写入磁盘的文件会放在path文件夹下,文件的名称是cache的名称,后缀名是data。  diskPersistent：是否持久化磁盘缓存,当这个属性的值为true时,系统在初始化时会在磁盘中查找文件名为cache名称,后缀名为index的文件，这个文件中存放了已经持久化在磁盘中的cache的index,找到后会把cache加载到内存，要想把cache真正持久化到磁盘,写程序时注意执行net.sf.ehcache.Cache.put(Element element)后要调用flush()方法。  diskExpiryThreadIntervalSeconds：磁盘缓存的清理线程运行间隔,默认是120秒。  diskSpoolBufferSizeMB：设置DiskStore（磁盘缓存）的缓存区大小,默认是30MB  memoryStoreEvictionPolicy：内存存储与释放策略,即达到maxElementsInMemory限制时,Ehcache会根据指定策略清理内存，共有三种策略,分别为LRU(最近最少使用)、LFU(最常用的)、FIFO(先进先出)。-->  <diskStore path="java.io.tmpdir/ehcache"/>  <!-- 默认缓存 -->  <defaultCache  maxElementsInMemory="1000"  eternal="false"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  overflowToDisk="false"/>    <!-- 菜单缓存 -->  <cache name="menuCache"  maxElementsInMemory="1000"  eternal="false"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  overflowToDisk="false"  memoryStoreEvictionPolicy="LRU"/>    </ehcache> |

#### application\_spring\_cache.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:cache="http://www.springframework.org/schema/cache"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd  http://www.springframework.org/schema/cache  http://www.springframework.org/schema/cache/spring-cache-3.2.xsd">    <cache:annotation-driven cache-manager="cacheManager"/>    <bean id="cacheManagerFactory" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">  <property name="configLocation" value="classpath:application/ehcache.xml" />  </bean>    <bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">  <property name="cacheManager" ref="cacheManagerFactory"/>  </bean>    </beans> |

#### 使用

首先在ehcache.xml中配置缓存策略，即添加一组cache。

1. 添加缓存

|  |
| --- |
| /\*  @Cacheable主要针对方法配置，能够根据方法的请求参数对其结果进行缓存。  Value: 缓存的名称，在 spring 配置文件中定义，必须指定至少一个  Key: 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合  condition: 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存  另外,还有一个@CachePut，主要针对方法配置，能够根据方法的请求参数对其结果进行缓存，和 @Cacheable 不同的是，它每次都会触发真实方法的调用，参数和cacheable一样。  \*/  @Cacheable(value="menuCache",key="'UserMenuKey'+#userid")  public List<MenuBean> queryMenuByUserId(String userid)  {  ……  } |

1. 删除缓存

|  |
| --- |
| /\*  @CachEvict主要针对方法配置，能够根据一定的条件对缓存进行清空。  Value: 缓存的名称，在 spring 配置文件中定义，必须指定至少一个  Key: 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合  condition: 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才清空缓存  allEntries：是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存  beforeInvocation：是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存  \*/  @CacheEvict(value="menuCache",key="'UserMenuKey'+#userRoleBean.getUserId()")  public boolean delUserRole(UserRoleBean userRoleBean) {  ……  } |

## 分布式配置

环境为两台机器

主机A ip：192.168.255.131

主机B ip：192.168.255.130

**1. RMI方式：**

rmi的方式配置要点（下面均是server1上的配置，server2上的只需要把ip兑换即可）

具体说明：配置cacheManagerPeerListenerFactory是配宿主主机配置监听程序，来发现其他主机发来的同步请求

配置cacheManagerPeerProviderFactory是指定除自身之外的网络群体中其他提供同步的主机列表，用“|”分开不同的主机。

下面的例子的测试过程是：主机B缓存开启，并从名为UserCache的缓存中循环抓取键值为“key1”的元素，直到取到，才退出循环。主机A缓存启动，并在名为UserCache的缓存中放入键值为“key1”的元素。显然，如果主机B取到的元素，那么就证明同步成功，也就是集群成功。

所以在测试过程中先启动主机B的测试程序，在启动主机A的测试程序。

注意：确保你所监听的端口已被服务器打开

下面具体说配置文件以及测试程序：

1. **主机A的配置文件以及测试源代码**

recluster\_ehcache.xml

|  |
| --- |
| <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="ehcache.xsd">  <cacheManagerPeerProviderFactory  class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"  properties="peerDiscovery=manual,rmiUrls=//192.168.255.130:40000/UserCache"/>    <cacheManagerPeerListenerFactory  class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"  properties="hostName=192.168.255.131,port=40000,socketTimeoutMillis=120000" />    <defaultCache maxElementsInMemory="10000" eternal="false"  timeToIdleSeconds="120" timeToLiveSeconds="120" overflowToDisk="true"  diskSpoolBufferSizeMB="30" maxElementsOnDisk="10000000"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120"  memoryStoreEvictionPolicy="LRU">  <cacheEventListenerFactory  class="net.sf.ehcache.distribution.RMICacheReplicatorFactory" />  </defaultCache>    <cache name="UserCache" maxElementsInMemory="1000" eternal="false"  timeToIdleSeconds="100000" timeToLiveSeconds="100000"  overflowToDisk="false">  <cacheEventListenerFactory  class="net.sf.ehcache.distribution.RMICacheReplicatorFactory" />  </cache>  </ehcache> |



Java代码

|  |
| --- |
| package ehcache;  import java.net.URL;  import net.sf.ehcache.Cache;  import net.sf.ehcache.CacheManager;  import net.sf.ehcache.Element;  public class clustertest {  /\*\*  \* @param args  \* @throws InterruptedException  \*/  public static void main(String[] args) throws InterruptedException{  // TODO Auto-generated method stub  //URL url = clustertest.class.getClassLoader().getResource("recluster\_ehcache.xml");  CacheManager manager = new CacheManager("recluster\_ehcache.xml");    //get Cache  Cache cache = manager.getCache("UserCache");  Element element = new Element("key1", "value1");  cache.put(element);  System.out.println("Initial:\n"//+url.toString()  +"\n"+manager.getName()  +"\n"+cache.getName()  +" 's size = "+cache.getSize()  +"\n"+element.toString());    Element element1 = cache.get("key1");  System.out.println(element1.getValue());  //while(true){  // Thread.sleep(1000);  //}  }  } |



1. **主机B上的配置文件以及测试代码**

recluster\_ehcache.xml

|  |
| --- |
| <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="ehcache.xsd">  <cacheManagerPeerProviderFactory  class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"  properties="peerDiscovery=manual,rmiUrls=//192.168.255.131:40000/UserCache" />  <cacheManagerPeerListenerFactory  class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"  properties="hostName=192.168.255.130,port=40000,socketTimeoutMillis=120000" />  <defaultCache maxElementsInMemory="10000" eternal="false"  timeToIdleSeconds="120" timeToLiveSeconds="120" overflowToDisk="true"  diskSpoolBufferSizeMB="30" maxElementsOnDisk="10000000"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120"  memoryStoreEvictionPolicy="LRU">  <cacheEventListenerFactory  class="net.sf.ehcache.distribution.RMICacheReplicatorFactory" />  </defaultCache>  <cache name="UserCache" maxElementsInMemory="1000" eternal="false"  timeToIdleSeconds="100000" timeToLiveSeconds="100000"  overflowToDisk="false">  <cacheEventListenerFactory  class="net.sf.ehcache.distribution.RMICacheReplicatorFactory" />  </cache>  </ehcache> |

Java代码

|  |
| --- |
| **package** ehcache;  **import** java.net.URL;  **import** net.sf.ehcache.CacheManager;  **import** net.sf.ehcache.Cache;  **import** net.sf.ehcache.Element;  **public** **class** cluster\_B {  /\*\*  \* **@param** args  \* **@throws** InterruptedException  \*/  **public** **static** **void** main(String[] args) **throws** InterruptedException {  // **TODO** Auto-generated method stub  //System.out.println("Test begin");  //URL url = cluster\_B.class.getClassLoader().getResource("recluster\_ehcache.xml");  CacheManager manager = **new** CacheManager("recluster\_ehcache.xml");  //System.out.println(url.toString());  //get Cache  Cache cache = manager.getCache("UserCache");    **int** i=0;  **while**(**true**) {    i++;  System.*out*.println(i+". "+cache.getName()+" 's size = "+cache.getSize());  Element e = cache.get("key1");  **if**(e != **null**) {  System.*out*.println(e.getValue());  **break**;  }  Thread.*sleep*(1000);  }  }  } |

主机B上成功取出value1即表示成功

**2. JGroups方式：**

ehcache 1.5.0之后版本支持的一种方式，配置起来比较简单，要点：

a. 配置PeerProvider，使用tcp的方式，例子如下：

|  |
| --- |
| <cacheManagerPeerProviderFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"  properties="connect=TCP(start\_port=7800):  TCPPING(initial\_hosts=192.168.2.154[7800],192.168.2.23[7800];port\_range=10;timeout=3000;  num\_initial\_members=3;up\_thread=true;down\_thread=true):  VERIFY\_SUSPECT(timeout=1500;down\_thread=false;up\_thread=false):  pbcast.NAKACK(down\_thread=true;up\_thread=true;gc\_lag=100;retransmit\_timeout=3000):  pbcast.GMS(join\_timeout=5000;join\_retry\_timeout=2000;shun=false;  print\_local\_addr=false;down\_thread=true;up\_thread=true)"  propertySeparator="::" /> |

 b.为每个cache添加cacheEventListener：

|  |
| --- |
| <cache name="userCache" maxElementsInMemory="10000" eternal="true"  overflowToDisk="true" timeToIdleSeconds="0" timeToLiveSeconds="0"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120">  <cacheEventListenerFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"  properties="replicateAsynchronously=true, replicatePuts=true,  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true"/>  </cache> |

 JGroup方式配置的两个server上的配置文件一样，若有多个server，在initial\_hosts中将server ip加上即可。

一个完整的ehcache.xml文件：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="http://ehcache.sf.net/ehcache.xsd">  <diskStore path="java.io.tmpdir" />  <cacheManagerPeerProviderFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"  properties="connect=TCP(start\_port=7800):  TCPPING(initial\_hosts=192.168.2.154[7800],192.168.2.23[7800];port\_range=10;timeout=3000;  num\_initial\_members=3;up\_thread=true;down\_thread=true):  VERIFY\_SUSPECT(timeout=1500;down\_thread=false;up\_thread=false):  pbcast.NAKACK(down\_thread=true;up\_thread=true;gc\_lag=100;retransmit\_timeout=3000):  pbcast.GMS(join\_timeout=5000;join\_retry\_timeout=2000;shun=false;  print\_local\_addr=false;down\_thread=true;up\_thread=true)"  propertySeparator="::" />  <defaultCache maxElementsInMemory="10000" eternal="true"  overflowToDisk="true" timeToIdleSeconds="0" timeToLiveSeconds="0"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120">  <cacheEventListenerFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"  properties="replicateAsynchronously=true, replicatePuts=true,  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true"/>  </defaultCache>  <cache name="velcroCache" maxElementsInMemory="10000" eternal="true"  overflowToDisk="true" timeToIdleSeconds="0" timeToLiveSeconds="0"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120">  <cacheEventListenerFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"  properties="replicateAsynchronously=true, replicatePuts=true,  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true"/>  </cache>  <cache name="userCache" maxElementsInMemory="10000" eternal="true"  overflowToDisk="true" timeToIdleSeconds="0" timeToLiveSeconds="0"  diskPersistent="false" diskExpiryThreadIntervalSeconds="120">  <cacheEventListenerFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"  properties="replicateAsynchronously=true, replicatePuts=true,  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true"/>  </cache>  <cache name="resourceCache" maxElementsInMemory="10000"  eternal="true" overflowToDisk="true" timeToIdleSeconds="0"  timeToLiveSeconds="0" diskPersistent="false"  diskExpiryThreadIntervalSeconds="120">  <cacheEventListenerFactory class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"  properties="replicateAsynchronously=true, replicatePuts=true,  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true"/>  </cache>  </ehcache> |

## 参考

<http://www.open-open.com/lib/view/open1322892011546.html>