**入门**

使用场景：Hibernate二级缓存、DAO对象、安全认证、Web Page等等。甚至可以用作SOAP和RESTFul服务端缓存。

集群之间的复制策略：除了自身提供的Terracotta外，还支持RMI、JGroups、JMS。

**主要的类和方法**

**CacheManager**：推荐使用单例模式创建，CacheManager.create()。

**Cache**：都实现了Ehcache接口。

**MemoryStore**、**DiskStore**：两种存储方式。

**Element**：被cache的元素。包含key、value和访问次数。对于不可序列化的Object，不能持久化和在集群间复制。

**性能考量**

尽可能多的使用堆内存，防止GC(垃圾回收)，如BigMemory(the off-heap store)。

**使用模式**

**cache-aside**：先查询缓存，能命中，则直接返回查询结果；否则，查询SOR(如db)，然后将结果缓存。(SOR：System-of-Record)

**cache-as-sor**：将缓存当做SOR使用。

**read-through**：读取数据的时候和cache-aside类似，不过需要实现CacheEntryFactory接口，指定缓存miss时如何读取数据(将显式变成隐式)。

**write-through**：写数据时和cache-aside类似，需要实现CacheWriter接口并将cache配置为write-through或者write-behind。

**write-behind**：启用另一个线程往SOR中写数据。

**动态配置**

timeToLive：最长存活时间(秒)，不管有没有被使用。默认值为0，永不过期。

timeToIdle：最长闲置时间(秒)，多少秒没有被使用就过期。默认值为0，永不过期。

Local Sizing Attributes：maxEntriesLocalHeap，maxBytesLocalOffHeap等等。

memory-store eviction policy。

CacheEventListeners添加和删除。

**BigMemory**

要配合Terracotta使用。它的原理是开辟一个不被JVM GC的heap，因此叫做off-heap。它以字节的方式存储对象，因此需要序列化。在xml配置中，有两个选项用于打开这个特性：overflowToOffHeap，表示是否启用；maxBytesLocalOffHeap，表示最大空间。同时，需要开启JVM选项支持：java -XX:MaxDirectMemorySize=2G

Memory Store --- off-heap Store --- Disk Store

**生命周期**

Flush：从上层刷出到下层，比如从Memory刷到Disk；

Fault：从下层copy到上层。

Eviction：从集群中移除cache实体。

Expiration：缓存过期。过期了的缓存并不马上被flush或者evict，只是表示不可用。

Pinning：强制数据保存在某个层。

**大块加载(Bulk Loading)**

比如在系统启动的时候加载权限块。API方面和普通cache保持一致，但添加了一些bulk方面的方法。

**Ehcache中的事务**

Local：保证cache中变化的原子性，操作只影响cache本身。

XA：Cache和[比如]DB之间的同步，非严格。

XA\_Strict：严格的XA事务，Cache和DB之间严格保持同步。

参与事务的cache对象必须实现Serializable接口并且覆盖equals和hashcode方法。在配置文件中，**transactionalMod**属性用于配置事务，可选值为：local，xa，xa\_strict和off，默认值是off。

与Spring事务集成。

**BlockingCache和SelfPopulatingCache**

BlockingCache场景：10个线程请求同一个动态数据，该动态数据很占资源，如果反复请求10次...假如这10个线程都阻塞，直到其中一个获取到动态数据并放入缓存，其他线程都从该缓存中读取，则可以大大提高系统的伸缩性。

SelfPopulatingCache场景：

**Event Listener、Cache扩展、Eviction策略**

CacheManagerEventListener和CacheEventListener，分别针对Cache和Element的变动。

Memory中的eviction：LRU(最近最少使用)、LFU(使用频率最少)和FIFO(先进先出)

磁盘中的eviction：LFU

**操作**

调整GC：使用jstat检测GC问题，如果GC太频繁或者GC时间太长，则明显会感觉程序停顿。jstat -gcutil <pid> 10 1000000。Sum给的GC策略：

java ... -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC

-XX:NewSize=<1/4 of total heap size> -XX:SurvivorRatio=16

**复制(Replication)**

使用RMI：

① 它是Java默认的远程工具

② 它已经成熟

③ 可以调节TCP选项

④ 使用RMI传输不需要再实现第三方转换，因为已经序列化

⑤ 可以穿透防火墙

要使用RMI复制，需要为CacheManager配置PeerProvider和CacheManagerPeerListener。在RMI中，每个节点都是一个peer，如何查找peer有两种方法，手动和自动。

手动，配置**cacheManagerPeerProviderFactory**属性为内建的net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory类。

自动，使用TCP多播机制形成一个Group。group使用心跳检测peer，每个peer每秒向group发送一个心跳，连续5秒未发送则断掉。想要成为这种peer，需要将cacheManagerPeerProviderFactory设置为下面例子中的字符串形式。

示例：

<cacheManagerPeerProviderFactory

class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"

properties="peerDiscovery=automatic, multicastGroupAddress=230.0.0.1,

multicastGroupPort=4446, timeToLive=32"/>

配置CacheManagerPeerListener，接收来自peer的信息。

<cacheManagerPeerListenerFactory

class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"

properties="hostName=localhost, port=40001,

socketTimeoutMillis=2000"/>

最后，为Cache配置需要复制的事件：

<cache name="sampleCache2"

maxEntriesLocalHeap="10"

eternal="false"

timeToIdleSeconds="100"

timeToLiveSeconds="100"

overflowToDisk="false">

<cacheEventListenerFactory

class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"

properties="replicateAsynchronously=true, replicatePuts=true, replicateUpdates=true,

replicateUpdatesViaCopy=false, replicateRemovals=true "/>

</cache>

使用JGroup：

将cacheManagerPeerProviderFactory改为JGroupsCacheManagerPeerProviderFactory，将cacheEventListenerFactory改为JGroupsCacheReplicatorFactory

使用JMS：比较复杂，参考Reference。

**JDBC**

1、标识出需要被缓存的方法

A、使用id访问数据的方法

B、可以忍受不一致和过期数据的方法

2、实例化一个cache对象，在DAO中持有这个cache对象的引用

在Spring环境下，为DAO增加getter/setter方法，在配置文件中将cache注入。将cache声明为bean进行管理

3、对cache进行put/get操作

使用相关的cache-aside来完成

示例代码参见reference第282页。

**Web Page & Fragment**

1、添加一个servlet的page cache filter到web.xml中。

2、在ehcache.xml中配置一个适当的cache供使用。

关键点：filter的名称和cache名称相同，从而关联起来。

**过期(Expiration)**

保持cache和SOR数据一致的几种方法：

1、过期策略如LRU等，配合timeToIdel和timeToLive等属性。

2、信息总线：类似JMS的发布、订阅。

3、触发器：数据库触发，不推荐。

**缓存空值(caching empty value)**

场景：数据库的查询结果本身为空，要避免频繁查询。此时可以将key和null值缓存。注意，这个null值是Element为null，不是value为null，两者之间的关系如下：

value = element.getObjectValue();

**并发读(Thundering Herd)**

场景：多个线程同时请求同一个数据，该数据尚未被缓存，需要从SOR如数据库中获取。

解决：使用BlockingCache，阻塞这些线程，只允许一个线程请求数据库，然后让其他线程访问回填后的缓存。

**Write-Behind**

当数据库接收到大量并发写操作时，会形成很高的负载。此时可以使用write-behind模式。该模式的思路是：当有写请求时，先更新缓存，将更新值放入一个队列，开启另一个后台线程将值更新到数据库。

问题：1、可能会有更新丢失，如突然断电、故障；2、事务处理变得复杂。

**代码示例**

创建CacheManager：

CacheManager.create();CacheManager.getInstance();cacheManager.newInstance();

可接受xml、URL、InputStream作为参数，指定加载的配置文件。

添加、获取、删除Cache：

cacheManager.addCache(“cache”); // 添加 cache必须添加到CacheManager才能使用

cacheManager.getCache(“cache”); // 获取

cacheManager.removeCache(“cache”); // 删除

Cache的使用：

Element e = cache.get(“”);

cache.put(“key”,e);

Object value = e.getObjectValue(); // 非Serializable的value

Serializable value = e.getValue();

cache.flush(); // 和Hibernate中的Session的flush方法相似

Cache的一些方法(相关配置参数)：

long elementsInMemory = cache.getMemoryStoreSize();

监控Cache

page 332