Ehcache 内存框架介绍

1. 背景及基础介绍

目的：使用缓存技术缓解数据库压力，提高访问速度，牺牲内存空间换取时间；

1.1 流行缓存框架简介

（1）OSCache OSCache 是个一个广泛采用的高性能的 J2EE 缓存框架，OSCache 能用于任何 Java 应用程序的普通的缓存解决方案。

OSCache 有以下特点：缓存任何对象，你可以不受限制的缓存部分 jsp 页面或 HTTP 请求，任何 java 对象都可以缓存。拥有全面的 API--OSCache API 给你全面的程序来控制所有的 OSCache 特性。永久缓存--缓存能随意的写入硬盘，因此允许昂贵的创建（expensive-to-create）数据来保持缓存，甚至能让应用重启。支持集群--集群缓存数据能被单个的进行参数配置，不需要修改代码。缓存记录的过期--你可以有最大限度的控制缓存对象的过期，包括可插入式的刷新策略（如果默认性能不需要时）。

（2）Java Caching system JSC(Java Caching system)是一个用分布式的缓存系统，是基于服务器的

java 应用程序。它是通过提供管理各种动态缓存数据来加速动态 web 应用。 JCS 和其他缓存系统一样，也是一个用于高速读取，低速写入的应用程序。动态内容和报表系统能够获得更好的性能。如果一个网站，有重复的网站结构，使用间歇性更新方式的数据库（而不是连续不断的更新数据库），被重复搜索出相同结果的，就能够通过执行缓存方式改进其性能和伸缩性。

（3）JCache JCache 是个开源程序，正在努力成为 JSR-107 开源规范，JSR-107 规范已经很多年没改变了。这个版本仍然是构建在最初的功能定义上。

（4）ShiftOne ShiftOne Java Object Cache 是一个执行一系列严格的对象缓存策略的 Java lib，就像一个轻量级的配置缓存工作状态的框架。

（5）SwarmCache SwarmCache 是一个简单且有效的分布式缓存，它使用 IP multicast 与同一个局域网的其他主机进行通讯，是特别为集群和数据驱动 web 应用程序而设计的。SwarmCache 能够让典型的读操作大大超过写操作的这类应用提供更好的性能支持。 SwarmCache 使用 JavaGroups 来管理从属关系和分布式缓存的通讯。

（6） TreeCache / JBossCache JBossCache 是一个复制的事务处理缓存，它允许你缓存企业级应用数据来更好的改善性能。缓存数据被自动复制，让你轻松进行 JBoss 服务器之间的集群工作。

JBossCache 能够通过 JBoss 应用服务或其他 J2EE 容器来运行一个 MBean 服务，当然，它也能独立运行。 JBossCache 包括两个模块：TreeCache 和 TreeCacheAOP。 TreeCache --是一个树形结构复制的事务处理缓存。 TreeCacheAOP --是一个“面向对象”缓存，它使用 AOP 来动态管理 POJO(Plain Old Java Objects) 注：AOP 是 OOP 的延续，是 Aspect Oriented Programming 的缩写，意思是面向方面编程。

（7）WhirlyCache Whirlycache 是一个快速的、可配置的、存在于内存中的对象的缓存。它能够通过缓存对象来加快网站或应用程序的速度，否则就必须通过查询数据库或其他代价较高的处理程序来建立。

1.2 Ehcache 的主要特性和集群方案

EHCache EHCache 是一个纯 java 的在进程中的缓存，是 Hibernate 中默认的 CacheProvider，最小的依赖性，全面的文档和测试，最新版本为 2.0.1。

缓存应用在多个领域并发挥作用，ehcache 可应用于数据库访问缓存，安全认证缓存，web 缓存，soap 和 RESTFul 服务缓存，应用程序持久对象缓存以及分布式缓存。

（1）EhCache 的主要特性有：

a）快速；
b）简单；
c）多种缓存策略；
d）缓存数据有两级：内存和磁盘，因此无需担心容量问题；
e）缓存数据会在虚拟机重启的过程中写入磁盘；
f）可以通过 RMI、可插入 API 等方式进行分布式缓存；
g）具有缓存和缓存管理器的侦听接口；
h）支持多缓存管理器实例，以及一个实例的多个缓存区域；
i）提供 Hibernate 的缓存实现；

（2）EhCache 从 1.7 版本后，支持五种集群方案，分别是：

a）Terracotta
b）RMI
c）JMS
d）JGroups
e）EhCache Server

1.3 Ehcache 的层次模型

Ehcache 的类层次模型主要为三层，最上层的是 CacheManager，他是操作 Ehcache 的入口。我们可以通过 CacheManager.getInstance()获得一个单子的 CacheManger，或者通过 CacheManger 的构造函数创建　一个新的 CacheManger。

每个 CacheManager 都管理着多个 Cache。而每个 Cache 都以一种类 Hash 的方式，关联着多个 Element。

Element（键值对）则是我们用于存放要缓存内容的地方。

1.4 Hibernate 的二级缓存策略

Hibernate 的二级缓存策略的一般过程如下

1）条件查询的时候，总是发出一条 select * from table_name where ....

（选择所有字段）这样的 SQL 语句查询数据库，一次获得所有的数据对象。

2） 把获得的所有数据对象根据 ID 放入到第二级缓存中。

3）当 Hibernate 根据 ID 访问数据对象的时候，首先从 Session 一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；查不到，再查询数据库，把结果按照 ID 放入到缓存。

4 ）删除、更新、增加数据的时候，同时更新缓存。

Hibernate 的二级缓存策略，是针对于 ID 查询的缓存策略，对于条件查询则毫无作用。

为此，Hi bernate 提供了针对条件查询的 Query Cache。

1.5 Ehcache 的三种清空策略

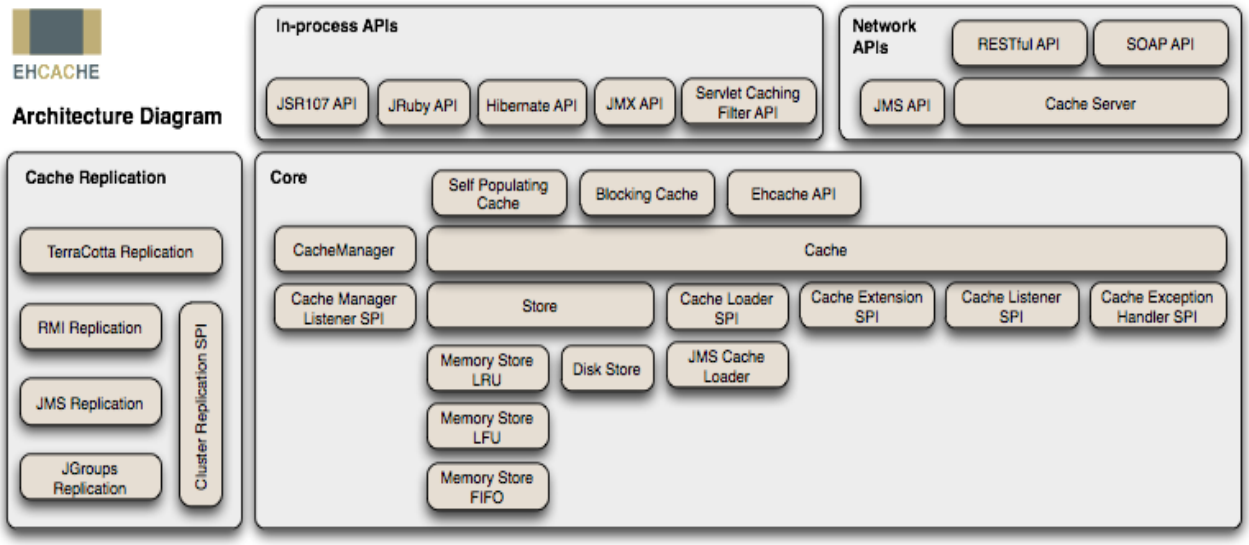
Ehcache 的三种清空策略

1 ： FIFO，first in first out，这个是大家最熟的，先进先出。

2 ： LFU， Less Frequently Used，就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个 hit 属性，hit 值最小的将会被清出缓存。

3 ： LRU，Least Recently Used，**最近最少使用的**，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。

1.6 EhCache 应用架构图



2. EhCache 配置介绍

1) ehcache-core-2.0.1.jar

2) ehcache.xml

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd" updateCheck="false">

<!-- 系统默认的 temp 文件目录 -->
    <diskStore path="java.io.tmpdir" />

    <!-- 多播（ multicast ）来维护集群中的所有有效节点;
    cacheManagerPeerProviderFactory：用于在 cluster 中查找 CacheManagers
    其中需要指定节点发现模式 peerDiscovery 值为 automatic 自动;
    同时组播地址可以指定 D 类 IP 地址空间，范围从 224.0.1.0 到 238.255.255.255 中的任何一个地址。
    cacheManagerPeerListenerFactory ： 指 定 CacheManagerPeerListenerFactory ， 用 于 创 建 CacheManagerPeerListener，监听 cluster 中的复制信息
    属性：
    -->
    <cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
    properties="peerDiscovery=automatic, multicastGroupAddress=231.1.0.8,
    multicastGroupPort=44467, timeToLive=32"
```

/>

<cacheManagerPeerListenerFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
properties="port=40011, remoteObjectPort=48097, socketTimeoutMillis=2000" />

<!-- 进行缓存　数据复制　的区域（Region）配置 ；
cache 子元素：
cacheEventListenerFactory：注册相应的的缓存监听类，用于处理缓存事件，如 put,remove,update,和 expire；
bootstrapCacheLoaderFactory:指定相应的 BootstrapCacheLoader，用于在初始化缓存，以及自动设置。

-->

<!—自定义 cache

<cache　name="outCache"
maxElementsInMemory="666"
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" />

<bootstrapCacheLoaderFactory

class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
properties="bootstrapAsynchronously=true" />

</cache>

-->

<!-- 默认 cache 的属性，创建一个新 cache 和此属性一致 -->
<defaultCache

maxElementsInMemory="809"
eternal="true"

```
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            overflowToDisk="true"
            diskSpoolBufferSizeMB="30"
            maxElementsOnDisk="10000000"
            diskPersistent="false"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU"      >

                                        <cacheEventListenerFactory
class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
                                            properties="replicateAsynchronously=true,
                                                                replicatePuts=true,
                                                                replicateUpdates=true,
                                                                replicateUpdatesViaCopy=false,
                                                                 replicateRemovals=true"       />

                                        <bootstrapCacheLoaderFactory

class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
                                                    properties="bootstrapAsynchronously=true" />

            </defaultCache>


    </ehcache>
```
f)

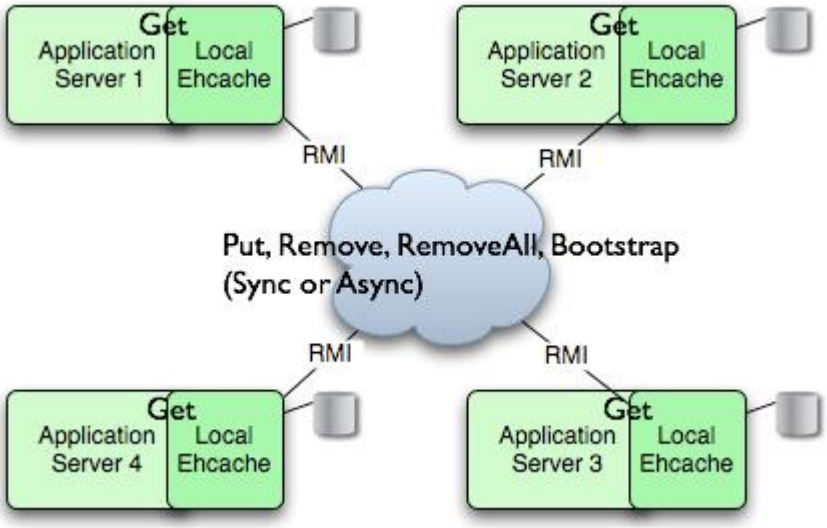
3. 代码示例和 API(开发演示)

4. EhCache RMI 集群方案介绍（见上面 RMI 配置）

1）RMI 是 Java 的一种远程方法调用技术，是一种点对点的基于 Java 对象的通讯方式。EhCache 从 1.2 版本开始就支持 RMI 方式的缓存集群。在集群环境中 EhCache 所有缓存对象的键和值都必须是可序列化的，也就是必须实现 java.io.Serializable 接口，这点在其它集群方式下也是需要遵守的。

采用 RMI 集群模式时，集群中的每个节点都是对等关系，并不存在主节点或者从节点的概念，因此节点间必须有一个机制能够互相认识对方，必须知道其它节点的信息，包括主机地址、端口号等。EhCache 提供两种节点的发现方式：手工配置和自动发现。手工配置方式要求在每个节点中配置其它所有节点的连接信息，一旦集群中的节点发生变化时，需要对缓存进行重新配置。

2）RMI 集群模式结构图：



5. EhCache JGroups 集群方案介绍

1）JGroups 集群模式

EhCache 从 1.5. 版本开始增加了 JGroups 的分布式集群模式。与 RMI 方式相比较， JGroups 提供了一个非常灵活的协议栈、可靠的单播和多播消息传输，主要的缺点是配置复杂以及一些协议栈对第三方包的依赖。

JGroups 也提供了基于 TCP 的单播（Unicast）和基于 UDP 的多播（Multicast），对应 RMI 的手工配置和自动发现。使用单播方式需要指定其它节点的主机地址和端口，下面是两个节点，并使用了

单播方式的配置：

```
<cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
    properties="connect=TCP(start_port=7800):
        TCPPING(initial_hosts=host1[7800],host2[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="::" />
```

使用多播方式配置如下：

```
<cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
    properties="connect=UDP(mcast_addr=231.12.21.132;mcast_port=45566;):PING:
    MERGE2:FD_SOCK:VERIFY_SUSPECT:pbcast.NAKACK:UNICAST:pbcast.STABLE:FRAG:pbcast.GMS"
    propertySeparator="::"
/>
```

2）从上面的配置来看，JGroups 的配置要比 RMI 复杂得多，但也提供更多的微调参数，有助于提升缓存数据复制的性能。详细可参考 JGroups 的配置手册。

JGroups 方式对应缓存节点的配置信息如下：

```
<cache name="sampleCache2"
    maxElementsInMemory="10"
    eternal="false"
    timeToIdleSeconds="100"
    timeToLiveSeconds="100"
    overflowToDisk="false">
    <cacheEventListenerFactory
        class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
        properties="replicateAsynchronously=true, replicatePuts=true,
        replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true" />
</cache>
```

3）使用组播方式的注意事项

使用 JGroups 需要引入 JGroups 的 Jar 包以及 EhCache 对 JGroups 的封装包 ehcache-jgroupsreplication-xxx.jar 。

在一些启用了 IPv6 的电脑中，经常启动的时候报如下错误信息：

java.lang.RuntimeException: the type of the stack (IPv6) and the user supplied addresses (IPv4) don't match: /231.12.21.132.

解决的办法是增加 JVM 参数：-Djava.net.preferIPv4Stack=true。如果是 Tomcat 服务器，可在 catalina.bat 或者 catalina.sh 中增加如下环境变量即可：

```
SET CATALINA_OPTS=-Djava.net.preferIPv4Stack=true
```

经过实际测试发现，集群方式下的缓存数据都可以在 1 秒钟之内完成到其节点的复制。

6. EhCache Server 介绍

1）Ehcache 已经有了一个 Cache Server，它以两种形式出现：适合大多数 Web 容器的 WAR 以及独立的服务器。Cache Server 有两种类型的 API：面向资源的 RESTful 以及 SOAP。这两种 API 都支持任何编程语言。EhCache Server 是一个独立的缓存服务器，其内部使用 EhCache 做为缓存系统，可利用前面提到的两种方式进行内部集群。

EhCache Server 同时也提供强大的安全机制、监控功能。在数据存储方面，最大的 Ehcache 单实例在内存中可以缓存 20GB。最大的磁盘可以缓存 100GB。通过将节点整合在一起，这样缓存数据就可以跨越节点，以此获得更大的容量。将缓存 20GB 的 50 个节点整合在一起就是 1TB 了。

2）应用方式：

第一种，也是最简单的方式就是装配运行着 Ehcache Server 的几个节点，然后让客户端根据对象的 Hashcode 来决定使用哪个 Server：

```
String[] cacheservers = new String[]{"cacheserver0.company.com", "cacheserver1.company.com",
"cacheserver2.company.com", "cacheserver3.company.com", "cacheserver4.company.com",
"cacheserver5.company.com"};
Object key = "123231";
int hash = Math.abs(key.hashCode());
int cacheserverIndex = hash % cacheservers.length;
String cacheserver =cacheservers[cacheserverIndex];
```

第二，一个负载均衡器（load balancer）来支持冗余，每个节点运行两个 Ehcache Server 实例，通过使用现有的分布式缓存方案（RMI 或者 JGroups）支持节点之间的数据复制。在这种方式下，客户端依旧使用 Hashcode 来决定使用哪个 Server，但是现在我们可以在负载均衡器所分配的虚拟 IP 后透明地处理失败。

第三种，方式就是转换职责以将请求路由给负载均衡器：

EhCache Server 的 RESTful 版基于 Jersey——JSR 311 参考实现。Jersey 的开发者之一 Paul Sandoz 谈到了如何使**用 Jersey 的客户端 API 以访问缓存来创建并得到一个示例 XML 文档：**

```
// retrieving a node
Node n = r.accept("application/xml").get(DOMSource.class).getNode();
// creating a node
String xmlDocument = "...";
Client c = Client.create();
WebResource r = c.resource(http://localhost:8080/ehcache/rest/sampleCache2/2);
r.type("application/xml").put(xmlDocument);
```

3）EhCache Server 提供的对缓存数据进行操作的方法：

OPTIONS /{cache}}

获取某个缓存的可用操作的信息。

HEAD /{cache}/{element}

获取缓存中某个元素的 HTTP 头信息，例如：

curl --head http://localhost:8080/ehcache/rest/sampleCache2/2

EhCache Server 返回的信息如下：

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: GlassFish/v3
Last-Modified: Sun, 27 Jul 2008 08:08:49 GMT
ETag: "1217146129490"
Content-Type: text/plain; charset=iso-8859-1
Content-Length: 157
Date: Sun, 27 Jul 2008 08:17:09 GMT
```

GET /{cache}/{element}

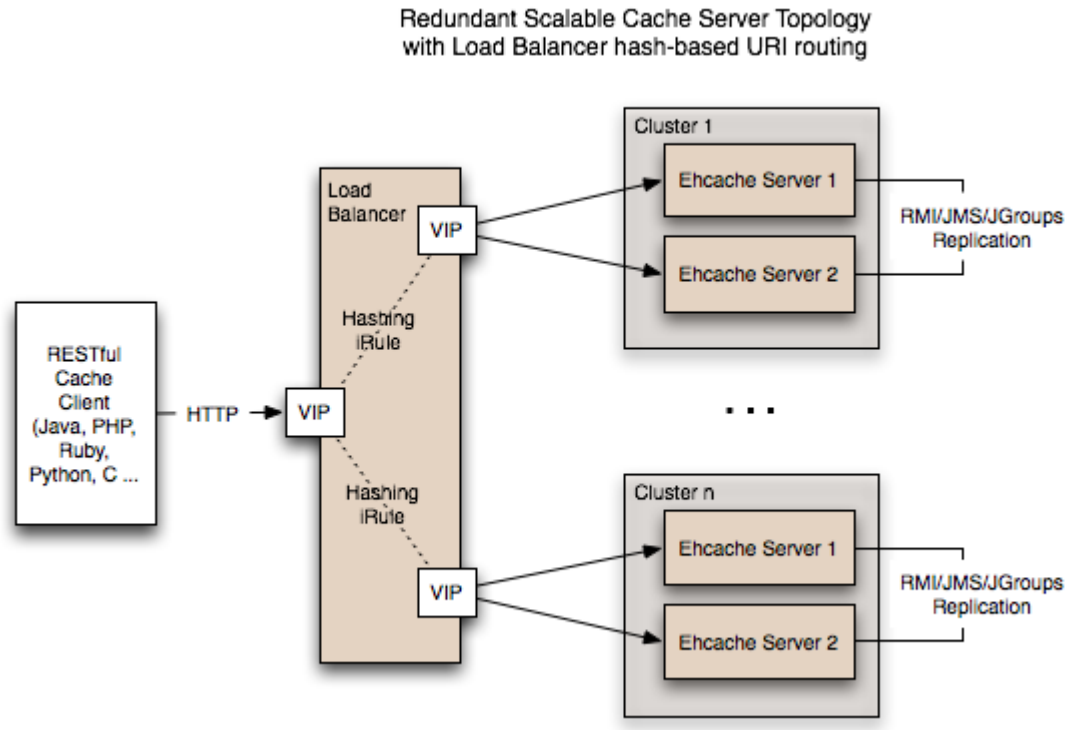
读取缓存中某个数据的值。

PUT /{cache}/{element}

写缓存。

由于这些操作都是基于 HTTP 协议的，因此你可以在任何一种编程语言中使用它，例如 Perl、PHP 和 Ruby 等等。

4）EhCache Server 应用架构图：



EhCache Server 是一个独立的缓存服务器，其内部使用 EhCache 做为缓存系统，可利用前面提到的两种方式进行内部集群。

7. Ehcache 和 MemCached 比较分析

项目	Memcache	Ehcache
分布式	不完全，集群默认不实现	支持
集群	可通过客户端实现	支持（默认是异步同步）
持久化	可通过第三方应用实现，如 sina 研发的 memcachedb，将 cache 的数据保存到	支持。持久化到本地硬盘，生成一个.data 和.index 文件。cache 初始化时会自动查找这

	[url=]Berkerly DB[/url]	两个文件，将数据放入 cache
效率	高	高于 Memcache
容灾	可通过客户端实现。	支持
缓存数据方式	缓存在 memcached server 向系统申请的内存中	可以缓存在内存（JVM 中），也可以缓存在硬盘。通过 CacheManager 管理 cache。多个 CacheManager 可配置在一个 JVM 内，CacheManager 可管理多个 cache。
缓存过期移除策略	LRU	[url=]LRU([/url]默认)，FIFO，LFU
缺点	功能不完善，相对于 Ehcache 效率低	只适用于 java 体系，只能用 java 编写客户端
优点	简洁，灵活，所有支持 socket 的语言都能编写其客户端	效率高。功能强大。

比较了一下 memcached 和 ehcache。
ehcache 是纯 java 编写的，通信是通过 RMI 方式，适用于基于 java 技术的项目。
memcached 服务器端是 c 编写的，
客户端有多个语言的实现，如 c，php（淘宝，sina 等各大门户网站），python（豆瓣网），java（Xmemcached，spymemcached)。memcached 服务器端是使用文本或者二进制通信的。
memcached 的 python 客户端没有开源，其他语言的好像都开源了。另外我以前不明白为什么各大互联网公司都是使用 memcached 缓存，后来我明白了原因：因为各大门户网站以及淘宝是使用 php 编写的网站，memcached 有 php 客户端，而 ehcache 是纯 java 的，SO.

memcache ： 是单独的服务器，服务器是 c 写的， 独立于 JVM（像个大 MAP）,客户端有 C、php、java（spymemcached）版本，一般不支持不同服务器间的 memcache 数据同步、不支持持久化；
一般存储重要的、读写频率高的重要的数据；---比如鉴权数据；

ehcache：是纯 java 的，支持不同服务器集群间的 cache 数据同步和持久化，一般存储只读的、配置类的数据；
ehcache 推荐使用 RMIcache 集群复制；