Ehcache缓存框架详解

一、前言

ehcache是一个比较成熟的 Java缓存框架，它提供了用内存，磁盘文件存储，以及分布式存储方式等多种灵活的 cache管理方案。 ehcache最早从 hibernate发展而来。由于 3.x的版本和 2.x的版本 API差异比较大。这里直接学习最新版本的了 ,但是最后整合 spring的时候还是有 2.x。

二、安装

由于我的项目是使用 maven管理的，因此只要在 pom文件中添加如下代码即可。

```
<dependency>
        <groupId>javax.cache</groupId>
        <artifactId>cache-api</artifactId>
        <version>1.0.0</version>
</dependency>
<dependency>
        <groupId>org.ehcache</groupId>
        <artifactId>ehcache</artifactId>
        <version>3.3.1</version>
</dependency>
```

好像 ehcache还要依赖上面的那个 Cache，因此最好两个多加上。

三、使用

1、快速入门 __JAVA实现

```
CacheManager cacheManager = CacheManagerBuilder.newCacheManagerBuilder()
        .withCache("preConfigured",
                CacheConfigurationBuilder.newCacheConfigurationBuilder(Long.class,          String.class,
ResourcePoolsBuilder.heap(10)))
        .build();
cacheManager.init();

Cache<Long, String> preConfigured =
        cacheManager.getCache("preConfigured", Long.class, String.class);

Cache<Long, String> myCache = cacheManager.createCache("myCache",
        CacheConfigurationBuilder.newCacheConfigurationBuilder(Long.class,                String.class,
ResourcePoolsBuilder.heap(10)));
```

```
myCache.put(1L, "da one!");
String value = myCache.get(1L);

cacheManager.removeCache("preConfigured");

cacheManager.close();
```

这里使用的官网的例子。

CacheManagerBuilder.newCacheManagerBuilder() ：创建缓存构造器的构建者， 类似工厂模式，注意，里面的实现不是单例。

.withCache() ：相当于创建一个自带缓存的 CacheManager。

build() ：这个方法就是创建 CacheManager 了。

上面的 build 方法接受一个 boolean 参数：当为 true 时， CacheManager 在使用时就不用初始化，否则，就需要 CacheManager 调用 init() 进行初始化操作。默认为 false。

cacheManager.getCache()：获取缓存

cacheManager.createCache()：创建缓存。

myCache.put(1L, “da one!”)：在缓存中添加值

2、快速入门 __XML 实现

```
<cache alias="foo">
        <!-- 缓存键值对的类型 -->
        <key-type>java.lang.String</key-type>
        <value-type>java.lang.String</value-type>
        <!-- 配置缓存 -->
        <expiry>
            <ttl unit="seconds">2</ttl>
        </expiry>
        <resources>
            <!-- 在堆中申请 2000 个 entries -->
            <heap unit="entries">2000</heap>
            <!-- 最大非堆内存 -->
            <offheap unit="MB">500</offheap>
        </resources>

</cache>
<!-- 定一个模板 -->
<cache-template name="myDefaults">
        <key-type>java.lang.Long</key-type>
        <value-type>java.lang.String</value-type>
        <heap unit="entries">200</heap>
</cache-template>
```

```
        <!--  使用上面的模板     -->
        <cache alias="bar" uses-template="myDefaults">
            <key-type>java.lang.Number</key-type>
        </cache>

        <cache alias="simpleCache" uses-template="myDefaults" />
```

```
// 测试 xml
    @Test
    public void test03() {
        URL url = getClass().getResource("/ehcache.xml");
        XmlConfiguration conf = new XmlConfiguration(url);
        CacheManager cacheManager = CacheManagerBuilder.newCacheManager(conf);
        cacheManager.init();
        Cache<String, String> cache = cacheManager.getCache("foo", String.class,
String.class);
        cache.put("key", "value");
        try {
            Thread.sleep(1000); //  测试过期时间
            String value = cache.get("key");
            System.out.println("result:" + value);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

上面即演示了 xml 实现缓存 ,注释都比较清楚了。之所以在 java 中调用 Thread.sleep(1000), 就是为了测试在 xml 中设置过期时间是否有效。

三、磁盘缓存

```
PersistentCacheManagerpersistentCacheManager=
CacheManagerBuilder.newCacheMawww.whfengjun.comnagerBuilder()
                .with(CacheManagerBuilder.persistence(new File("myData")))
                .withCache("threeTieredCache",

CacheConfigurationBuilder.newCacheConfigurationBuilder(Long.class, String.class,

ResourcePoolsBuilder.newResourcePoolsBuilder().heap(10, EntryUnit.ENTRIES)
                                .offheap(1, MemoryUnit.MB).disk(20,
MemoryUnit.MB, true)))
                .build(true);
```

Cache<Long, String> threeTieredCache = persistentCacheManager.getCache("threeTieredCache", Long.class, String.class);

threeTieredCache.put(1L, "stillAvailableAfterRestart");

persistentCacheManager.close();

上面代码即会在当前项目的同级目录下创建一个 mydata 目录。

四、 Spring 和 Ehcache 整合

Spring 本身并没有提供缓存的功能，但是却对市面上好多缓存框架提供了支持，即也支持 Ehcache。由于 Spring4 好像对 ehcache3.x 不是太支持，因此这里选用 2.x。

ehcache.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>
    <!-- 指定一个文件目录，当 EhCache 把数据写到硬盘上时，将把数据写到这个文件目录下 -->
    <diskStore path="java.io.tmpdir" />

    <!-- 设定缓存的默认数据过期策略 -->
    <defaultCache maxElemenwww.edu800.cntsInMemory="10000" eternal="false"
        overflowToDisk="true" timeToIdleSeconds="10" timeToLiveSeconds="20"
        diskPersistent="false" diskExpiryThreadIntervalSeconds="120" />
    <!--
        name：缓存名称

        maxElementsInMemory ：内存中最大缓存对象数

        maxElementsOnDisk ：硬盘中最大缓存对象数，若是 0 表示无穷大

        eternal：true 表示对象永不过期， 此时会忽略 timeToIdleSeconds 和 timeToLiveSeconds 属性，默认为 false

        overflowToDisk ： true 表示当内存缓存的对象数目达到了

        maxElementsInMemory 界限后， 会把溢出的对象写到硬盘缓存中。 注意： 如果缓存的对象要写入到硬盘中的话，则该对象必须实现了 Serializable 接口才行。

        diskSpoolBufferSizeMB ：磁盘缓存区大小，默认为 30MB 。每个 Cache 都应该有自己的一个缓存区。
```

diskPersistent：是否缓存虚拟机重启期数据，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 ：磁盘失效线程运行时间间隔，默认为 120 秒

timeToIdleSeconds： 设定允许对象处于空闲状态的最长时间，以秒为单位。当对象自从最近一次被访问后，如果处于空闲状态的时间超过了 timeToIdleSeconds 属性值，这个对象就会过期，EHCache 将把它从缓存中清空。只有当 eternal 属性为 false，该属性才有效。如果该属性值为 0，则表示对象可以无限期地处于空闲状态

timeToLiveSeconds ：设定对象允许存在于缓存中的最长时间，以秒为单位。当对象自从被存放到缓存中后，如果处于缓存中的时间超过了 timeToLiveSeconds 属性值，这个对象就会过期，EHCache 将把它从缓存中清除。只有当 eternal 属性为 false，该属性才有效。如果该属性值为 0，则表示对象可以无限期地存在于缓存中。timeToLiveSeconds 必须大于 timeToIdleSeconds 属性，才有意义

memoryStoreEvictionPolicy ：当达到 maxElementsInMemory 限制时，Ehcache 将会根据指定的策略去清理内存。可选策略有： LRU （最近最少使用，默认策略）、FIFO（先进先出）、LFU（最少访问次数）。

```
    -->
    <cache name="cacheTest"
           maxElementsInMemory="1000"
           eternal="false"
           overflowToDisk="true" timeToIdleSeconds="10" timeToLiveSeconds="20" />

</ehcache>
```

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>

    <!-- 自动扫描注解的 bean -->
    <context:component-scan base-package="com.lw.spring.ehcache" />

    <!-- 注解扫描 -->
    <cache:annotation-driven cache-manager="ehCacheCacheManager" />

    <!-- 实例 CacheManager -->
    <bean id="ehCacheCacheManager"
class="org.springframework.cache.ehcache.EhCacheCacheManager">
        <property name="cacheManager" ref="ehcache"></property>
```

```
        </bean>

        <!--  加载  ehcache 配置文件    -->
        <bean id="ehcawww.tt951.comche"
              class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
              <property name="configLocation" value="classpath:ehcache.xml"></property>
        </bean>

</beans>
```

创建一个 service 接口，并添加实现。

```
package com.lw.spring.ehcache;

import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

@Service
public class EhcacheServiceImpl implements EhcacheService {

        /**
        * @Cacheable
        *  表明所修饰的方法是可以缓存的：当第一次调用这个方法时,
        *  它的结果会被缓存下来，在缓存的有效时间内。如果每次缓存返回的结果都是一样
的，则不会执行方法体的代码
        *  以后访问这个方法都直接返回缓存结果，不再执行方法中的代码段。
        * @CachePut: 功能比 @Cacheable 更强，不仅缓存方法结果，还缓存代码段，每次都会
执行方法体的内容
        * @CacheEvict ：功能和 @Cacheable 功能相反， @CacheEvict 表明所修饰的方法是用来
删除失效或无用的缓存数据。
        * @CacheConfig ：与前面的缓存注解不同，这是一个类级别的注解。如果类的所有操
作都是缓存操作，你可以使用  @CacheConfig 来指定类，省去一些配置。
        * condition: 缓存的条件，可以为  null, 使用  spel 编写，只有返回  true 的情况下才进行缓
存
        *  这个方法会以  key 中的值作为缓存中的  key,返回结果作为缓存中的值进行缓存
         */
        @CachePut(value = "cacheTest", key = "#param",condition="1<2")
        public String getTime(Swww.nc630.comtring param) {
              System.out.println("getTime  方法调用了  ...");
              Long time = System.currentTimeMillis();
              return time.toString();
        }
}
```

测试：

```
package com.lw.spring.ehcache;

import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.AbstractJUnit4SpringContextTests;

@ContextConfiguration(locations={"classpath:application.xml"})
public class EhcacheTest    extends AbstractJUnit4SpringContextTests{

    @Autowired
    private EhcacheService ehcacheService;

    @Test
    public void getTimestampTest() throws InterruptedException{
        System.out.println(" 第一次调用：  " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));

        System.out.println(" 调用： " + ehcacheService.getTime("param"));
        Thread.sleep(11000);// 10s 之后缓存消失
        System.out.println(" 再过  11 秒之后调用：  " + ehcacheService.getTime("param"));
    }
}
```

上面就完成了 Spring 和 ehcache 的结合了。其实主要就是 ehcache 的核心类交给了 Spring。

上面还有几个注解，注释里也解释了。