[**Ehcache 整合Spring 使用页面、对象缓存**](http://www.cnblogs.com/hoojo/archive/2012/07/12/2587556.html)

Ehcache在很多项目中都出现过，用法也比较简单。一 般的加些配置就可以了，而且Ehcache可以对页面、对象、数据进行缓存，同时支持集群/分布式缓存。如果整合Spring、Hibernate也非常 的简单，Spring对Ehcache的支持也非常好。EHCache支持内存和磁盘的缓存，支持LRU、LFU和FIFO多种淘汰算法，支持分布式的 Cache，可以作为Hibernate的缓存插件。同时它也能提供基于Filter的Cache，该Filter可以缓存响应的内容并采用Gzip压缩 提高响应速度。

Email：hoojo\_@126.com

Blog：<http://blog.csdn.net/IBM_hoojo>

<http://hoojo.cnblogs.com/>

**一、准备工作**

如果你的系统中已经成功加入Spring、Hibernate；那么你就可以进入下面Ehcache的准备工作。

1、 下载jar包

Ehcache 对象、数据缓存：<http://ehcache.org/downloads/destination?name=ehcache-core-2.5.2-distribution.tar.gz&bucket=tcdistributions&file=ehcache-core-2.5.2-distribution.tar.gz>

Web页面缓存：<http://ehcache.org/downloads/destination?name=ehcache-web-2.0.4-distribution.tar.gz&bucket=tcdistributions&file=ehcache-web-2.0.4-distribution.tar.gz>

2、 需要添加如下jar包到lib目录下

ehcache-core-2.5.2.jar

ehcache-web-2.0.4.jar 主要针对页面缓存

3、 当前工程的src目录中加入配置文件

ehcache.xml

ehcache.xsd

这些配置文件在ehcache-core这个jar包中可以找到

**二、Ehcache基本用法**

CacheManager cacheManager = CacheManager.create();

// 或者

cacheManager = CacheManager.getInstance();

// 或者

cacheManager = CacheManager.create("/config/ehcache.xml");

// 或者

cacheManager = CacheManager.create("http://localhost:8080/test/ehcache.xml");

cacheManager = CacheManager.newInstance("/config/ehcache.xml");

// .......

// 获取ehcache配置文件中的一个cache

Cache sample = cacheManager.getCache("sample");

// 获取页面缓存

BlockingCache cache = new BlockingCache(cacheManager.getEhcache("SimplePageCachingFilter"));

// 添加数据到缓存中

Element element = new Element("key", "val");

sample.put(element);

// 获取缓存中的对象，注意添加到cache中对象要序列化 实现Serializable接口

Element result = sample.get("key");

// 删除缓存

sample.remove("key");

sample.removeAll();

// 获取缓存管理器中的缓存配置名称

for (String cacheName : cacheManager.getCacheNames()) {

System.out.println(cacheName);

}

// 获取所有的缓存对象

for (Object key : cache.getKeys()) {

System.out.println(key);

}

// 得到缓存中的对象数

cache.getSize();

// 得到缓存对象占用内存的大小

cache.getMemoryStoreSize();

// 得到缓存读取的命中次数

cache.getStatistics().getCacheHits();

// 得到缓存读取的错失次数

cache.getStatistics().getCacheMisses();

**三、页面缓存**

页面缓存主要用Filter过滤器对请求的url进行过滤，如果该url在缓存中出现。那么页面数据就从缓存对象中获取，并以gzip压缩后返回。 其速度是没有压缩缓存时速度的3-5倍，效率相当之高！其中页面缓存的过滤器有CachingFilter，一般要扩展filter或是自定义 Filter都继承该CachingFilter。

CachingFilter功能可以对HTTP响应的内容进行缓存。这种方式缓存数据的粒度比较粗，例如缓存整张页面。它的优点是使用简单、效率高，缺点是不够灵活，可重用程度不高。

EHCache使用SimplePageCachingFilter类实现Filter缓存。该类继承自CachingFilter，有默认产生 cache key的calculateKey()方法，该方法使用HTTP请求的URI和查询条件来组成key。也可以自己实现一个Filter，同样继承 CachingFilter类,然后覆写calculateKey()方法，生成自定义的key。

CachingFilter输出的数据会根据浏览器发送的Accept-Encoding头信息进行Gzip压缩。

在使用Gzip压缩时，需注意两个问题：

1. Filter在进行Gzip压缩时，采用系统默认编码，对于使用GBK编码的中文网页来说，需要将操作系统的语言设置为：zh\_CN.GBK，否则会出现乱码的问题。

2. 默认情况下CachingFilter会根据浏览器发送的请求头部所包含的Accept-Encoding参数值来判断是否进行Gzip压缩。虽然 IE6/7浏览器是支持Gzip压缩的，但是在发送请求的时候却不带该参数。为了对IE6/7也能进行Gzip压缩，可以通过继承 CachingFilter，实现自己的Filter，然后在具体的实现中覆写方法acceptsGzipEncoding。

具体实现参考：

protected boolean acceptsGzipEncoding(HttpServletRequest request) {

boolean ie6 = headerContains(request, "User-Agent", "MSIE 6.0");

boolean ie7 = headerContains(request, "User-Agent", "MSIE 7.0");

return acceptsEncoding(request, "gzip") || ie6 || ie7;

}

在ehcache.xml中加入如下配置

<?xml version="1.0" encoding="gbk"?>

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

<diskStore path="java.io.tmpdir"/>

<defaultCache maxElementsInMemory="10000" eternal="false" timeToIdleSeconds="30" timeToLiveSeconds="30" overflowToDisk="false"/>

<!--

配置自定义缓存

maxElementsInMemory：缓存中允许创建的最大对象数

eternal：缓存中对象是否为永久的，如果是，超时设置将被忽略，对象从不过期。

timeToIdleSeconds：缓存数据的钝化时间，也就是在一个元素消亡之前，

两次访问时间的最大时间间隔值，这只能在元素不是永久驻留时有效，

如果该值是 0 就意味着元素可以停顿无穷长的时间。

timeToLiveSeconds：缓存数据的生存时间，也就是一个元素从构建到消亡的最大时间间隔值，

这只能在元素不是永久驻留时有效，如果该值是0就意味着元素可以停顿无穷长的时间。

overflowToDisk：内存不足时，是否启用磁盘缓存。

memoryStoreEvictionPolicy：缓存满了之后的淘汰算法。

-->

<cache name="SimplePageCachingFilter"

maxElementsInMemory="10000"

eternal="false"

overflowToDisk="false"

timeToIdleSeconds="900"

timeToLiveSeconds="1800"

memoryStoreEvictionPolicy="LFU" />

</ehcache>

具体代码：

package com.hoo.ehcache.filter;

import java.util.Enumeration;

import javax.servlet.FilterChain;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import net.sf.ehcache.CacheException;

import net.sf.ehcache.constructs.blocking.LockTimeoutException;

import net.sf.ehcache.constructs.web.AlreadyCommittedException;

import net.sf.ehcache.constructs.web.AlreadyGzippedException;

import net.sf.ehcache.constructs.web.filter.FilterNonReentrantException;

import net.sf.ehcache.constructs.web.filter.SimplePageCachingFilter;

import org.apache.commons.lang.StringUtils;

import org.apache.log4j.Logger;

/\*\*

\* <b>function:</b> mobile 页面缓存过滤器

\* @author hoojo

\* @createDate 2012-7-4 上午09:34:30

\* @file PageEhCacheFilter.java

\* @package com.hoo.ehcache.filter

\* @project Ehcache

\* @blog http://blog.csdn.net/IBM\_hoojo

\* @email hoojo\_@126.com

\* @version 1.0

\*/

public class PageEhCacheFilter extends SimplePageCachingFilter {

private final static Logger log = Logger.getLogger(PageEhCacheFilter.class);

private final static String FILTER\_URL\_PATTERNS = "patterns";

private static String[] cacheURLs;

private void init() throws CacheException {

String patterns = filterConfig.getInitParameter(FILTER\_URL\_PATTERNS);

cacheURLs = StringUtils.split(patterns, ",");

}

@Override

protected void doFilter(final HttpServletRequest request,

final HttpServletResponse response, final FilterChain chain)

throws AlreadyGzippedException, AlreadyCommittedException,

FilterNonReentrantException, LockTimeoutException, Exception {

if (cacheURLs == null) {

init();

}

String url = request.getRequestURI();

boolean flag = false;

if (cacheURLs != null && cacheURLs.length > 0) {

for (String cacheURL : cacheURLs) {

if (url.contains(cacheURL.trim())) {

flag = true;

break;

}

}

}

// 如果包含我们要缓存的url 就缓存该页面，否则执行正常的页面转向

if (flag) {

String query = request.getQueryString();

if (query != null) {

query = "?" + query;

}

log.info("当前请求被缓存：" + url + query);

super.doFilter(request, response, chain);

} else {

chain.doFilter(request, response);

}

}

@SuppressWarnings("unchecked")

private boolean headerContains(final HttpServletRequest request, final String header, final String value) {

logRequestHeaders(request);

final Enumeration accepted = request.getHeaders(header);

while (accepted.hasMoreElements()) {

final String headerValue = (String) accepted.nextElement();

if (headerValue.indexOf(value) != -1) {

return true;

}

}

return false;

}

/\*\*

\* @see net.sf.ehcache.constructs.web.filter.Filter#acceptsGzipEncoding(javax.servlet.http.HttpServletRequest)

\* <b>function:</b> 兼容ie6/7 gzip压缩

\* @author hoojo

\* @createDate 2012-7-4 上午11:07:11

\*/

@Override

protected boolean acceptsGzipEncoding(HttpServletRequest request) {

boolean ie6 = headerContains(request, "User-Agent", "MSIE 6.0");

boolean ie7 = headerContains(request, "User-Agent", "MSIE 7.0");

return acceptsEncoding(request, "gzip") || ie6 || ie7;

}

}

这里的PageEhCacheFilter继承了SimplePageCachingFilter，一般情况下 SimplePageCachingFilter就够用了，这里是为了满足当前系统需求才做了覆盖操作。使用 SimplePageCachingFilter需要在web.xml中配置cacheName，cacheName默认是 SimplePageCachingFilter，对应ehcache.xml中的cache配置。

在web.xml中加入如下配置

<!-- 缓存、gzip压缩核心过滤器 -->

<filter>

<filter-name>PageEhCacheFilter</filter-name>

<filter-class>com.hoo.ehcache.filter.PageEhCacheFilter</filter-class>

<init-param>

<param-name>patterns</param-name>

<!-- 配置你需要缓存的url -->

<param-value>/cache.jsp, product.action, market.action </param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>PageEhCacheFilter</filter-name>

<url-pattern>\*.action</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>PageEhCacheFilter</filter-name>

<url-pattern>\*.jsp</url-pattern>

</filter-mapping>

当第一次请求这些页面后，这些页面就会被添加到缓存中，以后请求这些页面将会从缓存中获取。你可以在cache.jsp页面中用小脚本来测试该页面 是否被缓存。<%=new Date()%>如果时间是变动的，则表示该页面没有被缓存或是缓存已经过期，否则则是在缓存状态了。

**四、对象缓存**

对象缓存就是将查询的数据，添加到缓存中，下次再次查询的时候直接从缓存中获取，而不去数据库中查询。

对象缓存一般是针对方法、类而来的，结合Spring的Aop对象、方法缓存就很简单。这里需要用到切面编程，用到了Spring的MethodInterceptor或是用@Aspect。

代码如下：

package com.hoo.common.ehcache;

import java.io.Serializable;

import net.sf.ehcache.Cache;

import net.sf.ehcache.Element;

import org.aopalliance.intercept.MethodInterceptor;

import org.aopalliance.intercept.MethodInvocation;

import org.apache.log4j.Logger;

import org.springframework.beans.factory.InitializingBean;

/\*\*

\* <b>function:</b> 缓存方法拦截器核心代码

\* @author hoojo

\* @createDate 2012-7-2 下午06:05:34

\* @file MethodCacheInterceptor.java

\* @package com.hoo.common.ehcache

\* @project Ehcache

\* @blog http://blog.csdn.net/IBM\_hoojo

\* @email hoojo\_@126.com

\* @version 1.0

\*/

public class MethodCacheInterceptor implements MethodInterceptor, InitializingBean {

private static final Logger log = Logger.getLogger(MethodCacheInterceptor.class);

private Cache cache;

public void setCache(Cache cache) {

this.cache = cache;

}

public void afterPropertiesSet() throws Exception {

log.info(cache + " A cache is required. Use setCache(Cache) to provide one.");

}

public Object invoke(MethodInvocation invocation) throws Throwable {

String targetName = invocation.getThis().getClass().getName();

String methodName = invocation.getMethod().getName();

Object[] arguments = invocation.getArguments();

Object result;

String cacheKey = getCacheKey(targetName, methodName, arguments);

Element element = null;

synchronized (this) {

element = cache.get(cacheKey);

if (element == null) {

log.info(cacheKey + "加入到缓存： " + cache.getName());

// 调用实际的方法

result = invocation.proceed();

element = new Element(cacheKey, (Serializable) result);

cache.put(element);

} else {

log.info(cacheKey + "使用缓存： " + cache.getName());

}

}

return element.getValue();

}

/\*\*

\* <b>function:</b> 返回具体的方法全路径名称 参数

\* @author hoojo

\* @createDate 2012-7-2 下午06:12:39

\* @param targetName 全路径

\* @param methodName 方法名称

\* @param arguments 参数

\* @return 完整方法名称

\*/

private String getCacheKey(String targetName, String methodName, Object[] arguments) {

StringBuffer sb = new StringBuffer();

sb.append(targetName).append(".").append(methodName);

if ((arguments != null) && (arguments.length != 0)) {

for (int i = 0; i < arguments.length; i++) {

sb.append(".").append(arguments[i]);

}

}

return sb.toString();

}

}

这里的方法拦截器主要是对你要拦截的类的方法进行拦截，然后判断该方法的类路径+方法名称+参数值组合的cache key在缓存cache中是否存在。如果存在就从缓存中取出该对象，转换成我们要的返回类型。没有的话就把该方法返回的对象添加到缓存中即可。值得主意的 是当前方法的参数和返回值的对象类型需要序列化。

我们需要在src目录下添加applicationContext.xml完成对MethodCacheInterceptor拦截器的配置，该配置主意是注入我们的cache对象，哪个cache来管理对象缓存，然后哪些类、方法参与该拦截器的扫描。

添加配置如下：

<context:component-scan base-package="com.hoo.common.interceptor"/>

<!-- 配置eh缓存管理器 -->

<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>

<!-- 配置一个简单的缓存工厂bean对象 -->

<bean id="simpleCache" class="org.springframework.cache.ehcache.EhCacheFactoryBean">

<property name="cacheManager" ref="cacheManager" />

<!-- 使用缓存 关联ehcache.xml中的缓存配置 -->

<property name="cacheName" value="mobileCache" />

</bean>

<!-- 配置一个缓存拦截器对象，处理具体的缓存业务 -->

<bean id="methodCacheInterceptor" class="com. hoo.common.interceptor.MethodCacheInterceptor">

<property name="cache" ref="simpleCache"/>

</bean>

<!-- 参与缓存的切入点对象 (切入点对象，确定何时何地调用拦截器) -->

<bean id="methodCachePointCut" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

<!-- 配置缓存aop切面 -->

<property name="advice" ref="methodCacheInterceptor" />

<!-- 配置哪些方法参与缓存策略 -->

<!--

.表示符合任何单一字元

### +表示符合前一个字元一次或多次

### \*表示符合前一个字元零次或多次

### \Escape任何Regular expression使用到的符号

-->

<!-- .\*表示前面的前缀(包括包名) 表示print方法-->

<property name="patterns">

<list>

<value>com.hoo.rest.\*RestService\*\.\*get.\*</value>

<value>com.hoo.rest.\*RestService\*\.\*search.\*</value>

</list>

</property>

</bean>

在ehcache.xml中添加如下cache配置

<cache name="mobileCache"

maxElementsInMemory="10000"

eternal="false"

overflowToDisk="true"

timeToIdleSeconds="1800"

timeToLiveSeconds="3600"

memoryStoreEvictionPolicy="LFU" />