缓存使用场景：

·小的、经常使用的、很少修改的内容保存在缓存中；如：数据字典；

·当需要保存的用户特定信息，不能受用户登录退出影响时，可保存在缓存中；如：用户vip提醒；

实现1：

数据字典缓存；---应用启动时，保存到缓存中；

实现2：

用户vip提醒：

用户登录后，查询vip状态，过期的，每天首次登录后提醒一次；

~

实现逻辑：

用户登录后，据用户id为key，查询缓存中是否有该用户的vip提醒缓存；

vip提醒缓存中，获得时间值，判断是否该天的，是：说明改天已提醒过，则不提醒；

不是：重置缓存，返回需提示的结果；

重置缓存：key=用户id，value保存时间值；

~

见“缓存使用2”文档；

//////////////////////////////////////////////////////////////////////////////////////////////////////////

Spring,hibernate整合ehcache使用：

实现方式：

注解方式，或者xml-aop方式；

xml-aop方式，

Spring配置文件，

<bean id=*"cacheManager"* class=*"org.springframework.cache.ehcache.EhCacheCacheManager"* p:cache-manager-ref=*"ehcache"*/>

<!-- EhCache library setup -->

<bean id="ehcache" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean" p:config-location="ehcache.xml">

<!-- 由于hibernate也使用了Ehcache, 保证双方都使用同一个缓存管理器 -->

<property name="shared" value="true"/>

</bean>

<!-- cache definitions -->

<cache:advice id=*"cacheAdvice"* cache-manager=*"cacheManager"*>

<cache:caching cache=*"findAllCache"*>

<cache:cacheable method=*"findAll"* />

<cache:cache-evict method=*"reload"* all-entries=*"true"*/>

</cache:caching>

</cache:advice>

<!-- apply the cacheable behavior to all BookService interfaces -->

<aop:config>

<aop:advisor advice-ref=*"cacheAdvice"* pointcut=*"execution(\* com.neirong.cms.service.\*.\*(..))"*/>

</aop:config>

Resources文件夹下建立*ehcache.xml文件，*

*Ehcache.xml*，

<ehcache>

<cache name=*"findAllCache"*

eternal=*"true"*

timeToIdleSeconds=*"900"*

timeToLiveSeconds=*"1800"*

maxElementsInMemory=*"2000"*

overflowToDisk=*"false"*

memoryStoreEvictionPolicy=*"LRU"* />

</ehcache>

name 缓存名称；

eternal="true" 缓存对象是否永久有效，false时，下面两条设置生效，true时，下面两条不生效；

timeToIdleSeconds="900" 闲置失效时间；0为无穷； //单位秒

timeToLiveSeconds="1800" 存活时间；0为无穷；

maxElementsInMemory="1000" 内存缓存中最大存放数量，超过时：或者存放入磁盘缓存，或者替换已有

overflowToDisk="false" 内存不足时,是否启用磁盘缓存

memoryStoreEvictionPolicy=”LFU” 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，时间单位)、LFU(最常用的，清除点击最少的)、FIFO(先进先出)

~

maxElementsOnDisk 磁盘缓存中最多可以存放的元素数量,

**<diskStore** path="java.io.tmpdir"**/>** 磁盘缓存写入位置；

diskPersistent 是否持久化磁盘缓存

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

数据操作类，

一切照旧，不需要特别标识cache；

//////////////////////////////////////////////////////////////////////////////////////////////////////////

项目启动时，加载数据字典缓存；

spring配置文件中，

<bean id=*"dictionary"* class=*"com.cxdai.common.Dictionary"* init-method=*"init"* />

//查询数据字典内容，cache操作类；

**public** **class** Dictionary {

**private** **static** **final** Logger ***logger*** = Logger.*getLogger*(Dictionary.**class**);

**private** **static** Dictionary *\_THIS*;

@Autowired

**private** ConfigurationService configurationService;

@Autowired

**private** org.springframework.cache.CacheManager cacheManager;

**public** **void** init() {

*\_THIS* = **this**;

*\_THIS*.configurationService.queryAllConfigurations();

}

//通过类型获得对应数据字典内容，

**private** **static** **final** LinkedHashMap<String, Configuration> getConfigurationMap(Integer type) {

**try** {

**return** *\_THIS*.configurationService.queryAllConfigurations().get(type);

} **catch** (Exception e) {

***logger***.error("dictionary error", e);

**return** **null**;

}

}

**public** **static** **final** Collection<Configuration> getValues(Integer type) {

LinkedHashMap<String, Configuration> map = *getConfigurationMap*(type);

**if** (map != **null** && !map.isEmpty()) {

**return** map.values();

}

**return** Collections.*emptyList*();

}

//转换下参数，

**public** **static** **final** String getValue(String type, String name) {

**try** {

**return** *getValue*(Integer.*valueOf*(type), name);

} **catch** (Exception e) {

**return** "";

}

}

//类型---name--value，

**public** **static** **final** String getValue(Integer type, String name) {

LinkedHashMap<String, Configuration> map = *getConfigurationMap*(type);

**if** (map != **null** && !map.isEmpty()) {

Configuration configuration = map.get(name);

**if** (configuration != **null**) {

**return** configuration.getValue();

}

}

**return** "";

}

**public** **static** **void** cleanUp() {

*\_THIS*.cacheManager.getCache("queryAllConfigurations").clear();

}

}

//查询，并保存cache；

@Service

**public** **class** ConfigurationServiceImpl **implements** ConfigurationService {

@Autowired

**private** ConfigurationMapper configurationMapper;

@Cacheable(value = "queryAllConfigurations")

**public** HashMap<Integer, LinkedHashMap<String, Configuration>> queryAllConfigurations() {

List<Configuration> configurations = configurationMapper.selectAll();

HashMap<Integer, LinkedHashMap<String, Configuration>> ret = **new** HashMap<Integer, LinkedHashMap<String, Configuration>>();

Integer type = **null**;

**for** (Configuration configuration : configurations) {

**if** (!ret.containsKey(type = configuration.getType())) {

ret.put(type, **new** LinkedHashMap<String, Configuration>());

}

ret.get(type).put(configuration.getName(), configuration);

}

**return** ret;

}

}

或：

<http://www.cnblogs.com/modou/articles/1325806.html>

<http://blog.csdn.net/xiajun07061225/article/details/40211391>