EhCache

EhCache简介

官网：https://www.ehcache.org/

Ehcache是一个用Java实现的使用简单，高速，实现线程安全的缓存管理类库，ehcache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。同时ehcache作为开放源代码项目，采用限制比较宽松的Apache License V2.0作为授权方式，被广泛地用于Hibernate, Spring，Cocoon等其他开源系统。Ehcache 从 Hibernate 发展而来，逐渐涵盖了Cahce 界的全部功能,是目前发展势头最好的一个项目。具有快速,简单,低消耗，依赖性小，扩展

性强,支持对象或序列化缓存，支持缓存或元素的失效，提供 LRU、LFU 和 FIFO 缓存策略，支持内存缓存和磁盘缓存，分布式缓存机制等等特点。

缓存特征

根据面向对象的软件思维来看，缓存就是一个对象类型，那么必然有它的属性：

- **命中率**

命中率=返回正确结果数/请求缓存次数，命中率问题是缓存中的一个非常重要的问题，它是衡量缓存有效性的重要指标。命中率越高，表明缓存的使用率越高。

- **最大元素（或最大空间）**

缓存中可以存放的最大元素的数量，一旦缓存中元素数量超过这个值（或者缓存数据所占空间超过其最大支持空间），那么将会触发缓存启动清空策略根据不同的场景合理的设置最大元素值往往可以一定程度上提高缓存的命中率，从而更有效的时候缓存。

- **清空策略**

如上描述，缓存的存储空间有限制，当缓存空间被用满时，如何保证在稳定服务的同时有效提升命中率？这就由缓存清空策略来处理，设计适合自身数据特征的情况策略能有效提升命中率。常见的一般策略有：

a. FIFO（first in first out）

先进先出策略，最先进入缓存的数据在缓存空间不够的情况下（超出最大元素限制）会被优先被清除掉，以腾出新的空间接受新的数据。策略算法主要比较缓存元素的创建时间。

b. LFU（less frequently used）

最少使用策略，无论是否过期，根据元素的被使用次数判断，清除使用次数较少的元素释放空间。策略算法主要比较元素的hitCount（命中次数）。

c. LRU（least recently used）

最近最少使用策略，无论是否过期，根据元素最后一次被使用的时间戳，清除最远使用时间戳的元素释放空间。策略算法主要比较元素最近一次被get使用时间。

除此之外，还有一些简单策略比如：

根据过期时间判断，清理过期时间最长的元素；

根据过期时间判断，清理最近要过期的元素；

随机清理；

根据关键字（或元素内容）长短清理等。

缓存介质

（从硬件介质上来看，无非就是内存和硬盘两种）从技术上划分，可以分成几种，内存、硬盘文件、数据库。

- 内存：将缓存存储于内存中是最快的选择，无需额外的I/O开销，但是内存的缺点是没有持久化落地物理磁盘，一旦应用异常break down，重新启动数据很难或者无法复原。

- 硬盘：一般来说，很多缓存框架会结合使用内存和硬盘，在内存分配空间满了或是在异常的情况下，可以被动或主动的将内存空间数据持久化到硬盘中，达到释放空间或备份数据的目的。
- 数据库：前面我们有提到，增加缓存的策略的目的之一就是为了减少数据库的I/O压力。现在使用数据库做缓存介质是不是又回到了老问题上了？其实，数据库也有很多种类型，像那些不支持SQL，只是简单的key、value的存储结构的特殊数据库（如berkleydb），响应速度和吞吐量都远远高于我们常用的关系型数据库等。

在目前的应用服务框架中，我们对缓存的分类划分更常用的是根据缓存与应用的耦合程度，划分为local cache（本地缓存）和remote cache（分布式缓存）：

- Local cache：指的是在应用中的缓存组件，其最大的优点是应用和cache是在同一个进程内部，请求缓存非常快速，没有过多的网络开销等，在单应用不需要集群支持或者集群情况下各节点无需互相通知的场景下使用本地缓存较合适；同时，它的缺点也是应为缓存跟应用程序耦合，多个应用程序无法直接的共享缓存，各应用或集群的各节点都需要维护自己的单独缓存，对内存是一种浪费。
- Remote cache:：指的是与应用分离的缓存组件或服务，其最大的优点是自身就是一个独立的应用，与本地应用隔离，多个应用可直接的共享缓存。

目前各种类型的缓存都活跃在成千上万的应用服务中，还没有一种缓存方案可以解决一切的业务场景或数据类型，我们需要根据自身的特殊场景和背景，选择最适合的缓存方案。缓存的使用是程序员、架构师的必备技能，好的程序员能根据数据类型、业务场景来准确判断使用何种类型的缓存，如何使用这种缓存，以最小的成本最快的效率达到最优的目的。

主要特性：

- 快速，针对大型高并发系统场景，ehcache的多线程机制有相应的优化改善。
- 简单，很小的jar包，简单配置就可直接使用，单机场景下无需过多的其他服务依赖。
- 支持多种的缓存策略，灵活。
- 缓存数据有两级：内存和磁盘，与一般的本地内存缓存相比，有了磁盘的存储空间，将可以支持更大量的数据缓存需求。
- 具有缓存和缓存管理器的侦听接口，能更简单方便的进行缓存实例的监控管理。
- 支持多缓存管理器实例，以及一个实例的多个缓存区域。

ehcache 和 redis 比较

ehcache直接在jvm虚拟机中缓存，速度快，效率高；但是缓存共享麻烦，集群分布式应用不方便。

redis是通过socket访问到缓存服务，效率比Ehcache低，比数据库要快很多，处理集群和分布式缓存方便，有成熟的方案。如果是单个应用或者对缓存访问要求很高的应用，用ehcache。如果是大型系统，存在缓存共享、分布式部署、缓存内容很大的，建议用redis。

ehcache也有缓存共享方案，不过是通过RMI或者Jgroup多播方式进行广播缓存通知更新，缓存共享复杂，维护不方便；简单的共享可以，但是涉及到缓存恢复，大数据缓存，则不合适。

EhCache快速入门

EhCache2.x版本

使用步骤

第一步：在pom.xml中引入依赖

XML 复制代码

```xml
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.10.2</version>
</dependency>
```

第二步：在src/main/resources/创建一个配置文件 ehcache.xml

> 默认情况下Ehcache会自动加载classpath根目录下名为ehcache.xml文件，也可以将该文件放到其他地方在使用时指定文件的位置

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">

  <!-- 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/ehcache"/>

  <!-- 默认缓存 -->
  <!-- 设定缓存的默认数据过期策略 -->
  <defaultCache
          maxElementsInMemory="10000"
          eternal="false"
          timeToIdleSeconds="120"
          timeToLiveSeconds="120"
          maxElementsOnDisk="10000000"
          diskExpiryThreadIntervalSeconds="120"
          memoryStoreEvictionPolicy="LRU">
  </defaultCache>

  <!--
    设定具体的命名缓存的数据过期策略
    cache元素的属性：
        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
        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）。
  -->

```

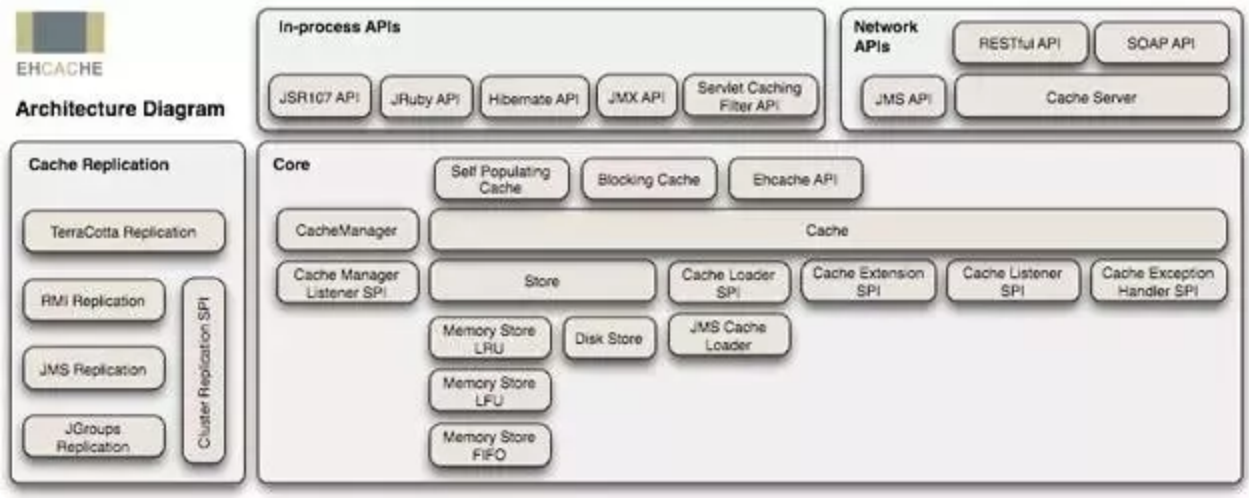
```
36
37
38      <!-- helloworld缓存 -->
39      <cache name="HelloWorldCache"
40             maxElementsInMemory="1000"
41             eternal="false"
42             timeToIdleSeconds="5"
43             timeToLiveSeconds="5"
44             overflowToDisk="false"
45             memoryStoreEvictionPolicy="LRU"/>
46    </ehcache>
```

第三步：编写测试类

Java 复制代码

```java
import entity.Dog;
import net.sf.ehcache.Cache;
import net.sf.ehcache.CacheManager;
import net.sf.ehcache.Element;

public class CacheTest {
    public static void main(String[] args) {
        // 1. 创建缓存管理器
        CacheManager cacheManager =
CacheManager.create("./src/main/resources/ehcache.xml");

        // 2. 获取缓存对象
        Cache cache = cacheManager.getCache("HelloWorldCache");

        // 3. 创建元素
        Element element = new Element("key1", "value1");

        // 4. 将元素添加到缓存
        cache.put(element);

        // 5. 获取缓存
        Element value = cache.get("key1");
        System.out.println("value: " + value);
        System.out.println(value.getObjectValue());

        // 6. 删除元素
        cache.remove("key1");

        Dog dog = new Dog("xiaohei", "black", 2);
        Element element2 = new Element("dog", dog);
        cache.put(element2);
        Element value2 = cache.get("dog");
        System.out.println("value2: "  + value2);
        Dog dog2 = (Dog) value2.getObjectValue();
        System.out.println(dog2);

        System.out.println(cache.getSize());

        // 7. 刷新缓存：将数据写入本地磁盘
        cache.flush();

        // 8. 关闭缓存管理器
        cacheManager.shutdown();

    }
}
```

案例小结



ehcache框架图

Ehcache API

- CacheManager：Cache的容器对象，并管理着（添加或删除）Cache的生命周期。

Java 复制代码

```
// 可以自己创建一个Cache对象添加到CacheManager中
public void addCache(Cache cache);
public synchronized void removeCache(String cacheName);
```

- Cache: 一个Cache可以包含多个Element，并被CacheManager管理。它实现了对缓存的逻辑行为
- Element：需要缓存的元素，它维护着一个键值对， 元素也可以设置有效期，0代表无限制

获取CacheManager的方式：

可以通过create()或者newInstance()方法或重载方法来创建获取CacheManager的方式：

Java 复制代码

```
public static CacheManager create();
public static CacheManager create(String configurationFileName);
public static CacheManager create(InputStream inputStream);
public static CacheManager create(URL configurationFileURL);

public static CacheManager newInstance();
```

Ehcache的CacheManager构造函数或工厂方法被调用时，会默认加载classpath下名为ehcache.xml的配置文件。

如果加载失败，会加载Ehcache jar包中的ehcache-failsafe.xml文件，这个文件中含有简单的默认配置。

```java
// CacheManager.create() == CacheManager.create("./src/main/resources/ehcache.xml")
// 使用Ehcache默认配置新建一个CacheManager实例
CacheManager cacheManager = CacheManager.create();
cacheManager = CacheManager.newInstance();

cacheManager = CacheManager.newInstance("./src/main/resources/ehcache.xml");

InputStream inputStream = new FileInputStream(new File("./src/main/resources/ehcache.xml"));
cacheManager = CacheManager.newInstance(inputStream);

String[] cacheNames = cacheManager.getCacheNames();  // [HelloWorldCache]
```

整体上看，ehcache的使用还是相对简单便捷的，提供了完整的各类API接口。需要注意的是，虽然ehcache支持磁盘的持久化，但是由于存在两级缓存介质，在一级内存中的缓存，如果没有主动的刷入磁盘持久化的话，在应用异常down机等情形下，依然会有缓存数据丢失的出现，为此可以根据需要将缓存刷到磁盘，将缓存条目刷到磁盘的操作可以通过cache.flush()方法来执行，需要注意的是，对于对象的磁盘写入，前提是要将对象进行序列化。

EhCache3.x版本【了解】

使用步骤

第一步：添加依赖

```xml
<dependency>
  <groupId>org.ehcache</groupId>
  <artifactId>ehcache</artifactId>
  <version>3.9.6</version>
</dependency>
```

第二步：编写测试类

Java 复制代码

```java
package com.chenj;


import org.ehcache.Cache;
import org.ehcache.CacheManager;
import org.ehcache.config.builders.CacheConfigurationBuilder;
import org.ehcache.config.builders.CacheManagerBuilder;
import org.ehcache.config.builders.ResourcePoolsBuilder;

public class TestEhCache3 {
    public static void main(String[] args) {
        //使用CacheManagerBuilder来创建一个预配置（pre-configured)缓存。
        /*
        第一个参数是一个别名，用于Cache和Cachemanager进行配合。
        第二个参数是org.ehcache.config.CacheConfiguration主要用来配置Cache。
        我们使用org.ehcache.config.builders.CacheConfigurationBuilder的静态方法newCacheConfigurationBuilder来创建一个默认配置实例。
         */
        CacheManager cacheManager =
CacheManagerBuilder.newCacheManagerBuilder()
                .withCache("preConfigured",

 CacheConfigurationBuilder.newCacheConfigurationBuilder(Long.class,
String.class,
                                ResourcePoolsBuilder.heap(100))
                                .build())
                .build(true);//在你开始使用CacheManager的时候，需要使用init()方法进行初始化。
        //cacheManager.init();

        //我们能取回在第二步中设定的pre-configured别名，我们对于key和要传递的值类型，要求是类型安全的，否则将抛出ClassCastException异常
        Cache<Long, String> preConfigured
                = cacheManager.getCache("preConfigured", Long.class,
String.class);
        preConfigured.put(1L,"wowowo");

        System.out.println("preConfigured"+preConfigured);
        //通过CacheManager创建出新的Cache
        Cache<Long, String> myCache = cacheManager.createCache("myCache",

 CacheConfigurationBuilder.newCacheConfigurationBuilder(Long.class,
String.class,
                        ResourcePoolsBuilder.heap(100)).build());
        //使用put方法存储数据
        myCache.put(1L, "da one!");
        //使用get方法获取数据
        String value = myCache.get(1L);
```

```
40
41              System.out.println("value:"+value);
42              //close方法将释放CacheManager所管理的缓存资源
43              cacheManager.close();
44          }
45      }
46
```

案例小结

1、静态方法CacheManagerBuilder.newCacheManagerBuilder将返回一个新的org.ehcache.config.builders.CacheManagerBuilder的实例。

2、当我们要构建一个缓存管理器的时候，使用CacheManagerBuilder来创建一个预配置（pre-configured)缓存。

- 第一个参数是一个别名，用于Cache和Cachemanager进行配合。
- 第二个参数是org.ehcache.config.CacheConfiguration主要用来配置Cache。我们使用org.ehcache.config.builders.CacheConfigurationBuilder的静态方法newCacheConfigurationBuilder来创建一个默认配置实例。

3、最后调用.build方法返回一个完整的实例，当然我们也能使用CacheManager来初始化。

4、在你开始使用CacheManager的时候，需要使用init()方法进行初始化。

5、我们能取回在第二步中设定的pre-configured别名，我们对于key和要传递的值类型，要求是类型安全的，否则将抛出ClassCastException异常。

6、可以根据需求，通过CacheManager创建出新的Cache。实例化和完整实例化的Cache将通过CacheManager.getCache API返回。

7、使用put方法存储数据。

8、使用get方法获取数据。

9、我们可以通过CacheManager.removeCache方法来获取Cache，但是Cache取出来以后CacheManager将会删除自身保存的Cache实例。

10、close方法将释放CacheManager所管理的缓存资源。

Spring整合EhCache

添加Spring相关的依赖

第一步：引入maven依赖

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.chenj</groupId>
    <artifactId>ehcacheDemo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <junit.version>4.10</junit.version>
        <spring.version>4.2.3.RELEASE</spring.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>net.sf.ehcache</groupId>
            <artifactId>ehcache</artifactId>
            <version>2.10.2</version>
        </dependency>


        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- springframework -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
```

```
48          <dependency>
49              <groupId>org.springframework</groupId>
50              <artifactId>spring-context</artifactId>
51              <version>${spring.version}</version>
52          </dependency>
53          <dependency>
54              <groupId>org.springframework</groupId>
55              <artifactId>spring-context-support</artifactId>
56              <version>${spring.version}</version>
57          </dependency>
58      </dependencies>
59
60  </project>
```

第二步：在src/main/resources添加ehcache.xml

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="https://www.ehcache.org/ehcache.xsd">
    <!-- 磁盘缓存位置 -->
    <diskStore path="java.io.tmpdir/ehcache"/>

    <!-- 默认缓存 -->
    <defaultCache
            maxEntriesLocalHeap="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxEntriesLocalDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </defaultCache>


    <!-- helloworld缓存 -->
    <cache name="HelloWorldCache"
           maxElementsInMemory="1000"
           eternal="false"
           timeToIdleSeconds="5"
           timeToLiveSeconds="5"
           overflowToDisk="false"
           memoryStoreEvictionPolicy="LRU"/>

    <cache name="UserCache"
           maxElementsInMemory="1000"
           eternal="false"
           timeToIdleSeconds="1800"
           timeToLiveSeconds="1800"
           overflowToDisk="false"
           memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```

第三步：添加spring配置文件applicationContext.xml、applicationContext-ehcache.xml

applicationContext.xml

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">

    <context:component-scan base-package="com.chenj.*"/>
</beans>
```

新建Spring整合EhCache配置文件

applicationContext-ehcache.xml

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/cache
        http://www.springframework.org/schema/cache/spring-cache-3.2.xsd">

    <description>ehcache缓存配置管理文件</description>

    <!-- 启用缓存注解开关 -->
    <cache:annotation-driven cache-manager="cacheManager"/>

    <bean id="cacheManager"
class="org.springframework.cache.ehcache.EhCacheCacheManager">
        <property name="cacheManager" ref="ehcache"/>
    </bean>

    <bean id="ehcache"
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
        <property name="configLocation" value="classpath:ehcache.xml"/>
    </bean>

</beans>
```

第四步：创建EhcacheService、EhcacheServiceImpl

EhcacheService.java

Java 复制代码

```java
package com.chenj.service;

import com.chenj.entity.User;

public interface EhcacheService {
    // 测试失效情况，有效期为5秒
    public String getTimestamp(String param);

    public String getDataFromDB(String key);

    public void removeDataAtDB(String key);

    public String refreshData(String key);


    public User findById(String userId);

    public boolean isReserved(String userId);

    public void removeUser(String userId);

    public void removeAllUser();
}

```

注解基本使用方法

Spring对缓存的支持类似于对事务的支持。

首先使用注解标记方法，相当于定义了切点，然后使用Aop技术在这个方法的调用前、调用后获取方法的入参和返回值，进而实现了缓存的逻辑。

@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，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。

这个注解可以用condition属性来设置条件，如果不满足条件，就不使用缓存能力，直接执行方法。

可以使用key属性来指定key的生成规则。

@Cacheable 支持如下几个参数：

value：缓存位置名称，不能为空，如果使用EHCache，就是ehcache.xml中声明的cache的name，指明将值缓存到哪个Cache中

key：默认情况下，缓存的 key 就是方法的参数，缓存的 value 就是方法的返回值

支持SpEL，如果要引用参数值使用井号加参数名，如：#userId，一般来说，我们的更新操作只需要刷新缓存中某一个值，所以定义缓存的key值的方式就很重要，最好是能够唯一，因为这样可以准确的清除掉特定的缓存，而不会影响到其它缓存值 ，本例子中使用实体加冒号再加ID组合成键的名称，如"user:1"、"order:223123"等

当有多个参数时，默认就使用多个参数来做 key ， 如果只需要其中某一个参数做 key ， 则可以在 @Cacheable 注解中，通过 key 属性来指定 key ， 如上代码就表示只使用 id 作为缓存的 key ， 如果对 key 有复杂的要求，可以自定义 keyGenerator 。
方式如下：

Java 复制代码

```java
@Component("selfKeyGenerate")
public  class SelfKeyGenerate implements KeyGenerator {
    @Override
    public Object generate(Object target, Method method, Object... params)
{
        return target.getClass().getSimpleName() + "#" + method.getName()
+ "(" + JSON.toJSONString(params) + ")";
    }
}
```

然后在使用的地方，利用注解中的`keyGenerator`来指定key生成策略

condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持SpEL

```java
// 将缓存保存到名称为UserCache中，键为"user:"字符串加上userId值，如 'user:1'
@Cacheable(value="UserCache", key="'user:' + #userId")
public User findById(String userId) {
    return (User) new User("1", "mengdee");
}

@Cacheable(value="UserCache", keyGenerator = "selfKeyGenerate")
public User findById(String userId) {
    return (User) new User("1", "mengdee");
}

// 将缓存保存进UserCache中，并当参数userId的长度小于12时才保存进缓存，
@Cacheable(value="UserCache", condition="#userId.length() < 12")
public boolean isReserved(String userId) {
    System.out.println("UserCache:"+userId);
    return false;
}
```

@CachePut

与@Cacheable不同，@CachePut不仅会缓存方法的结果，还会执行方法的代码段。它支持的属性和用法都与@Cacheable一致。

@CacheEvict

与@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。

@CacheEvict 支持如下几个参数：

value：缓存位置名称，不能为空，同上

key：缓存的key，默认为空，同上

condition：触发条件，只有满足条件的情况才会清除缓存，默认为空，支持SpEL

allEntries：true表示清除value中的全部缓存，默认为false

```java
//清除掉UserCache中某个指定key的缓存
@CacheEvict(value="UserCache",key="'user:' + #userId")
public void removeUser(User user) {
    System.out.println("UserCache"+user.getUserId());
}

//清除掉UserCache中全部的缓存
@CacheEvict(value="UserCache", allEntries=true)
public final void setReservedUsers(String[] reservedUsers) {
   System.out.println("UserCache deleteall");
}
```

EhcacheServiceImpl.java

Java 复制代码

```java
package com.chenj.service.impl;

import com.chenj.entity.User;
import com.chenj.service.EhcacheService;
import org.springframework.stereotype.Service;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
@Service
public class EhcacheServiceImpl implements EhcacheService {
    // value的值和ehcache.xml中的配置保持一致
    @Cacheable(value="HelloWorldCache", key="#param")
    public String getTimestamp(String param) {
        Long timestamp = System.currentTimeMillis();
        return timestamp.toString();
    }

    @Cacheable(value="HelloWorldCache", key="#key")
    public String getDataFromDB(String key) {
        System.out.println("从数据库中获取数据...");
        return key + ":" +
String.valueOf(Math.round(Math.random()*1000000));
    }

    @CacheEvict(value="HelloWorldCache", key="#key")
    public void removeDataAtDB(String key) {
        System.out.println("从数据库中删除数据");
    }

    @CachePut(value="HelloWorldCache", key="#key")
    public String refreshData(String key) {
        System.out.println("模拟从数据库中加载数据");
        return key + "::" +
String.valueOf(Math.round(Math.random()*1000000));
    }

    // ----------------------------------------------------------------
-----
    @Cacheable(value="UserCache", key="'user:' + #userId")
    public User findById(String userId) {
        System.out.println("模拟从数据库中查询数据");
        return new User(1, "mengdee");
    }

    @Cacheable(value="UserCache", condition="#userId.length()<12")
    public boolean isReserved(String userId) {
        System.out.println("UserCache:"+userId);
        return false;
```

```
46        }
47
48        //清除掉UserCache中某个指定key的缓存
49        @CacheEvict(value="UserCache",key="'user:' + #userId")
50        public void removeUser(String userId) {
51            System.out.println("UserCache remove:"+ userId);
52        }
53
54        //allEntries：true表示清除value中的全部缓存，默认为false
55        //清除掉UserCache中全部的缓存
56        @CacheEvict(value="UserCache", allEntries=true)
57        public void removeAllUser() {
58            System.out.println("UserCache delete all");
59        }
60    }
61
```

User.java

Java 复制代码

```java
package com.chenj.entity;

public class User {
    private int id;
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public User() {
    }

    public User(int id, String name) {
        super();
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + "]";
    }
}

```

第五步：编写测试类

Java 复制代码

```java
package com.chenj;

import com.chenj.service.EhcacheService;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.AbstractJUnit4SpringContextTests;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;


@ContextConfiguration(locations = {"classpath:spring/applicationContext.xml","classpath:spring/applicationContext-ehcache.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class TestSpringEhCache {
    @Autowired //@Autowired 是通过 byType 的方式去注入的， 使用该注解，要求接口只能有一个实现类。
    private EhcacheService ehcacheService;

    // 有效时间是5秒，第一次和第二次获取的值是一样的，因第三次是5秒之后所以会获取新的值
    @Test
    public void testTimestamp() throws InterruptedException{
        System.out.println("第一次调用: " + ehcacheService.getTimestamp("param"));
        Thread.sleep(2000);
        System.out.println("2秒之后调用: " + ehcacheService.getTimestamp("param"));
        Thread.sleep(4000);
        System.out.println("再过4秒之后调用: " + ehcacheService.getTimestamp("param"));
    }

    @Test
    public void testCache(){
        String key = "zhangsan";
        String value = ehcacheService.getDataFromDB(key); // 从数据库中获取数据...
        ehcacheService.getDataFromDB(key);  // 从缓存中获取数据，所以不执行该方法体
        ehcacheService.removeDataAtDB(key); // 从数据库中删除数据
        ehcacheService.getDataFromDB(key);  // 从数据库中获取数据...（缓存数据删除了，所以要重新获取，执行方法体）
    }

    @Test
    public void testPut(){
```

```
39          String key = "mengdee";
40          ehcacheService.refreshData(key);  // 模拟从数据库中加载数据
41          String data = ehcacheService.getDataFromDB(key);
42          System.out.println("data:" + data); // data:mengdee::103385
43
44          ehcacheService.refreshData(key);  // 模拟从数据库中加载数据
45          String data2 = ehcacheService.getDataFromDB(key);
46          System.out.println("data2:" + data2);   // data2:mengdee::180538
47      }
48
49
50      @Test
51      public void testFindById(){
52          ehcacheService.findById("2"); // 模拟从数据库中查询数据
53          ehcacheService.findById("2");
54      }
55
56      @Test
57      public void testIsReserved(){
58          ehcacheService.isReserved("123");
59          ehcacheService.isReserved("123");
60      }
61
62      @Test
63      public void testRemoveUser(){
64          // 线添加到缓存
65          ehcacheService.findById("1");
66
67          // 再删除
68          ehcacheService.removeUser("1");
69
70          // 如果不存在会执行方法体
71          ehcacheService.findById("1");
72      }
73
74      @Test
75      public void testRemoveAllUser(){
76          ehcacheService.findById("1");
77          ehcacheService.findById("2");
78
79          ehcacheService.removeAllUser();
80
81          ehcacheService.findById("1");
82          ehcacheService.findById("2");
83
84  //      模拟从数据库中查询数据
85  //      模拟从数据库中查询数据
86  //      UserCache delete all
87  //      模拟从数据库中查询数据
88  //      模拟从数据库中查询数据
```

```
89        }
90    }
91
92
```

SpringBoot整合EhCache

添加springboot相关的依赖

首先，来创建一个 Spring Boot 项目，引入 Cache 依赖

XML 复制代码

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

在 pom.xml 文件中，引入 Ehcache 依赖：

XML 复制代码

```xml
<dependency>
  <groupId>net.sf.ehcache</groupId>
  <artifactId>ehcache</artifactId>
  <version>2.10.2</version>
</dependency>
```

完整的pom.xml

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.chenj</groupId>
    <artifactId>spring-boot-ehcache</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-boot-ehcache</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-cache</artifactId>
        </dependency>
        <dependency>
            <groupId>net.sf.ehcache</groupId>
            <artifactId>ehcache</artifactId>
            <version>2.10.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
```

```
47                </plugin>
48            </plugins>
49        </build>
50
51    </project>
52
```

在 resources 目录下，添加 ehcache 的配置文件 ehcache.xml ， 文件内容如下：

XML 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="https://www.ehcache.org/ehcache.xsd">
    <!-- 磁盘缓存位置 -->
    <diskStore path="java.io.tmpdir/ehcache"/>

    <!-- 默认缓存 -->
    <defaultCache
            maxEntriesLocalHeap="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxEntriesLocalDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </defaultCache>


    <!-- helloworld缓存 -->
    <cache name="HelloWorldCache"
           maxElementsInMemory="1000"
           eternal="false"
           timeToIdleSeconds="5"
           timeToLiveSeconds="5"
           overflowToDisk="false"
           memoryStoreEvictionPolicy="LRU"/>

    <cache name="UserCache"
           maxElementsInMemory="1000"
           eternal="false"
           timeToIdleSeconds="1800"
           timeToLiveSeconds="1800"
           overflowToDisk="false"
           memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```

注意

默认情况下，这个文件名是固定的，必须叫 ehcache.xml ，如果一定要换一个名字，那么需要在 application.properties 中明确指定配置文件名，配置方式如下：

Plain Text 复制代码

```
spring.cache.ehcache.config=classpath:xxx.xml
```

开启缓存

开启缓存的方式，也和 Redis 中一样，如下添加 `@EnableCaching` 依赖即可：

Java 复制代码

```java
package com.chenj.springbootehcache;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;

@SpringBootApplication
@EnableCaching
public class SpringBootEhcacheApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootEhcacheApplication.class, args);
    }

}

```

使用缓存

创建EhcacheService、EhcacheServiceImpl

EhcacheService.java

Java 复制代码

```java
package com.chenj.service;

import com.chenj.entity.User;

public interface EhcacheService {
    // 测试失效情况，有效期为5秒
    public String getTimestamp(String param);

    public String getDataFromDB(String key);

    public void removeDataAtDB(String key);

    public String refreshData(String key);


    public User findById(String userId);

    public boolean isReserved(String userId);

    public void removeUser(String userId);

    public void removeAllUser();
}

```

EhcacheServiceImpl.java

Java 复制代码

```java
package com.chenj.service.impl;

import com.chenj.entity.User;
import com.chenj.service.EhcacheService;
import org.springframework.stereotype.Service;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
@Service
public class EhcacheServiceImpl implements EhcacheService {
    // value的值和ehcache.xml中的配置保持一致
    @Cacheable(value="HelloWorldCache", key="#param")
    public String getTimestamp(String param) {
        Long timestamp = System.currentTimeMillis();
        return timestamp.toString();
    }

    @Cacheable(value="HelloWorldCache", key="#key")
    public String getDataFromDB(String key) {
        System.out.println("从数据库中获取数据...");
        return key + ":" +
String.valueOf(Math.round(Math.random()*1000000));
    }

    @CacheEvict(value="HelloWorldCache", key="#key")
    public void removeDataAtDB(String key) {
        System.out.println("从数据库中删除数据");
    }

    @CachePut(value="HelloWorldCache", key="#key")
    public String refreshData(String key) {
        System.out.println("模拟从数据库中加载数据");
        return key + "::" +
String.valueOf(Math.round(Math.random()*1000000));
    }

    // ----------------------------------------------------------------
-----
    @Cacheable(value="UserCache", key="'user:' + #userId")
    public User findById(String userId) {
        System.out.println("模拟从数据库中查询数据");
        return new User(1, "mengdee");
    }

    @Cacheable(value="UserCache", condition="#userId.length()<12")
    public boolean isReserved(String userId) {
        System.out.println("UserCache:"+userId);
        return false;
```

```
46        }
47
48        //清除掉UserCache中某个指定key的缓存
49        @CacheEvict(value="UserCache",key="'user:' + #userId")
50        public void removeUser(String userId) {
51            System.out.println("UserCache remove:"+ userId);
52        }
53
54        //allEntries：true表示清除value中的全部缓存，默认为false
55        //清除掉UserCache中全部的缓存
56        @CacheEvict(value="UserCache", allEntries=true)
57        public void removeAllUser() {
58            System.out.println("UserCache delete all");
59        }
60    }
61
```

User.java

Java 复制代码

```java
package com.chenj.entity;

public class User {
    private int id;
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public User() {
    }

    public User(int id, String name) {
        super();
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + "]";
    }
}

```

编写测试类

Java 复制代码

```java
package com.chenj.springbootehcache;

import com.chenj.springbootehcache.service.EhcacheService;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SpringBootEhcacheApplicationTests {

    @Test
    void contextLoads() {
    }


    @Autowired //@Autowired 是通过 byType 的方式去注入的， 使用该注解，要求接口只能有一个实现类。
    private EhcacheService ehcacheService;

    // 有效时间是5秒，第一次和第二次获取的值是一样的，因第三次是5秒之后所以会获取新的值
    @Test
    public void testTimestamp() throws InterruptedException{
        System.out.println("第一次调用：" + ehcacheService.getTimestamp("param"));
        Thread.sleep(2000);
        System.out.println("2秒之后调用：" + ehcacheService.getTimestamp("param"));
        Thread.sleep(4000);
        System.out.println("再过4秒之后调用：" + ehcacheService.getTimestamp("param"));
    }

    @Test
    public void testCache(){
        String key = "zhangsan";
        String value = ehcacheService.getDataFromDB(key); // 从数据库中获取数据...
        ehcacheService.getDataFromDB(key);  // 从缓存中获取数据，所以不执行该方法体
        ehcacheService.removeDataAtDB(key); // 从数据库中删除数据
        ehcacheService.getDataFromDB(key);  // 从数据库中获取数据...（缓存数据删除了，所以要重新获取，执行方法体）
    }

    @Test
    public void testPut(){
        String key = "mengdee";
        ehcacheService.refreshData(key);  // 模拟从数据库中加载数据
```

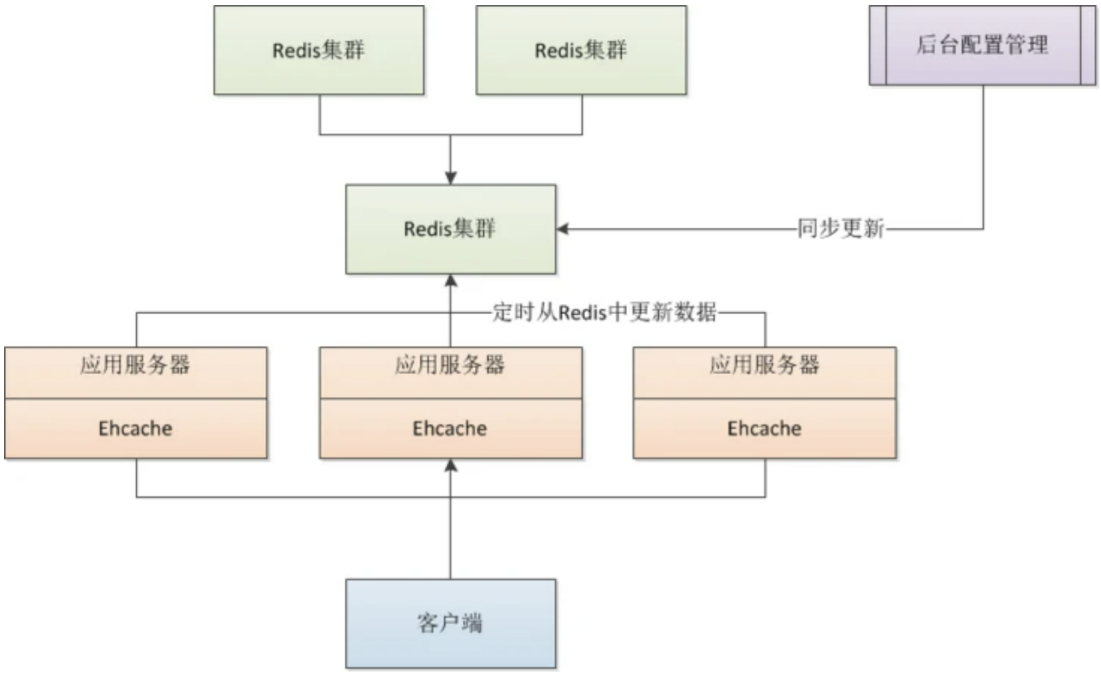
```
42          String data = ehcacheService.getDataFromDB(key);
43          System.out.println("data:" + data); // data:mengdee::103385
44
45          ehcacheService.refreshData(key);  // 模拟从数据库中加载数据
46          String data2 = ehcacheService.getDataFromDB(key);
47          System.out.println("data2:" + data2);   // data2:mengdee::180538
48      }
49
50
51      @Test
52      public void testFindById(){
53          ehcacheService.findById("2"); // 模拟从数据库中查询数据
54          ehcacheService.findById("2");
55      }
56
57      @Test
58      public void testIsReserved(){
59          ehcacheService.isReserved("123");
60          ehcacheService.isReserved("123");
61      }
62
63      @Test
64      public void testRemoveUser(){
65          // 线添加到缓存
66          ehcacheService.findById("1");
67
68          // 再删除
69          ehcacheService.removeUser("1");
70
71          // 如果不存在会执行方法体
72          ehcacheService.findById("1");
73      }
74
75      @Test
76      public void testRemoveAllUser(){
77          ehcacheService.findById("1");
78          ehcacheService.findById("2");
79
80          ehcacheService.removeAllUser();
81
82          ehcacheService.findById("1");
83          ehcacheService.findById("2");
84
85  //      模拟从数据库中查询数据
86  //      模拟从数据库中查询数据
87  //      UserCache delete all
88  //      模拟从数据库中查询数据
89  //      模拟从数据库中查询数据
90      }
91
```

```
92    }
93
```

实际工作中如何使用Ehcache

在实际工作中，我更多是将Ehcache作为与Redis配合的二级缓存。

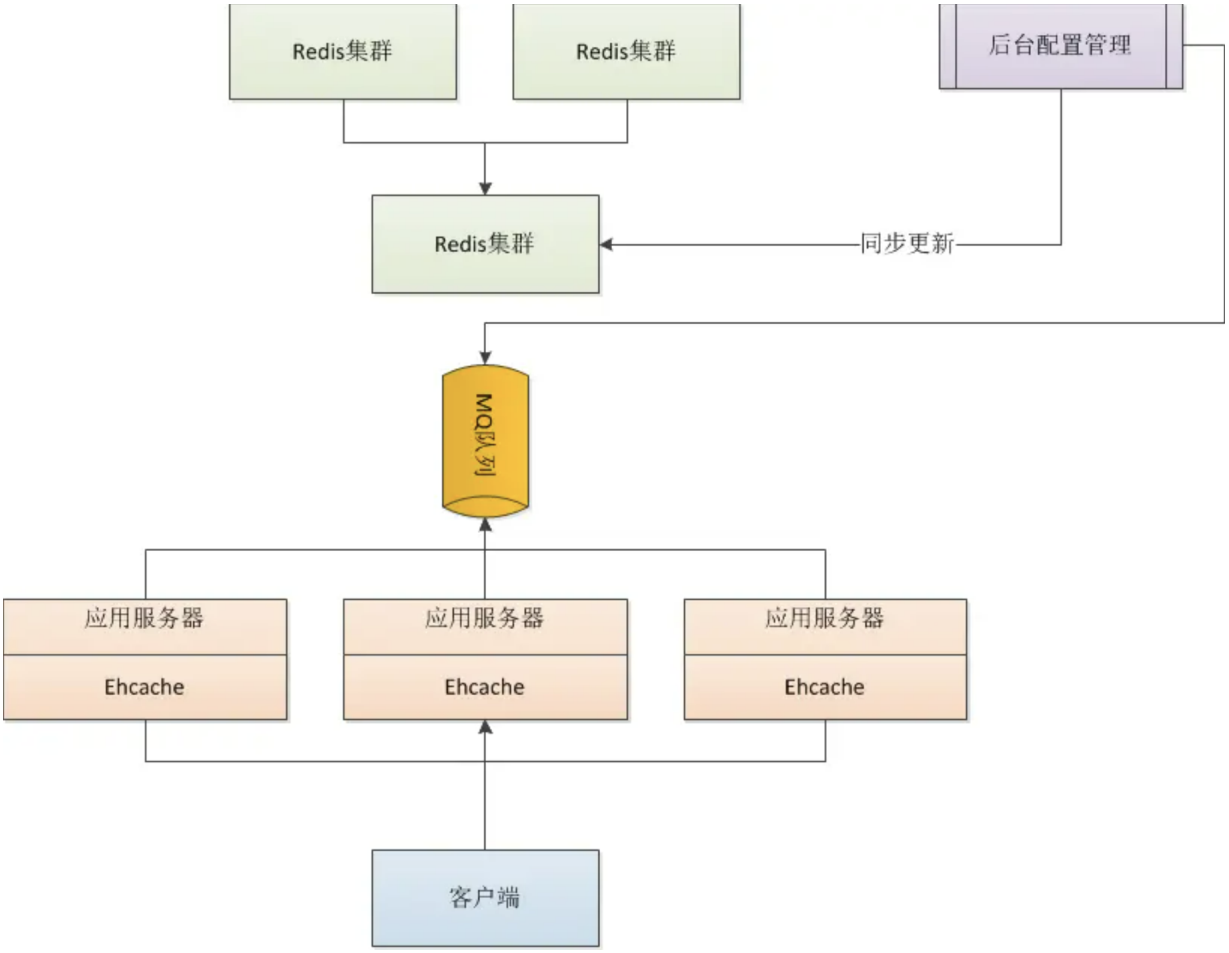
第一种方式：



注：

这种方式通过应用服务器的Ehcache定时轮询Redis缓存服务器更同步更新本地缓存，缺点是因为每台服务器定时Ehcache的时间不一样，那么不同服务器刷新最新缓存的时间也不一样，会产生数据不一致问题，对一致性要求不高可以使用。

第二种方式：



注：

通过引入了MQ队列，使每台应用服务器的Ehcache同步侦听MQ消息，这样在一定程度上可以达到**准同步**更新数据，通过MQ推送或者拉取的方式，但是因为不同服务器之间的网络速度的原因，所以也不能完全达到强一致性。基于此原理使用Zookeeper等分布式协调通知组件也是如此。

总结：

1、使用二级缓存的好处是减少缓存数据的网络传输开销，当集中式缓存出现故障的时候，Ehcache等本地缓存依然能够支撑应用程序正常使用，增加了程序的健壮性。另外使用二级缓存策略可以在一定程度上阻止缓存穿透问题。

2、根据CAP原理我们可以知道，如果要使用强一致性缓存（根据自身业务决定），集中式缓存是最佳选择，如（Redis，Memcached等）。