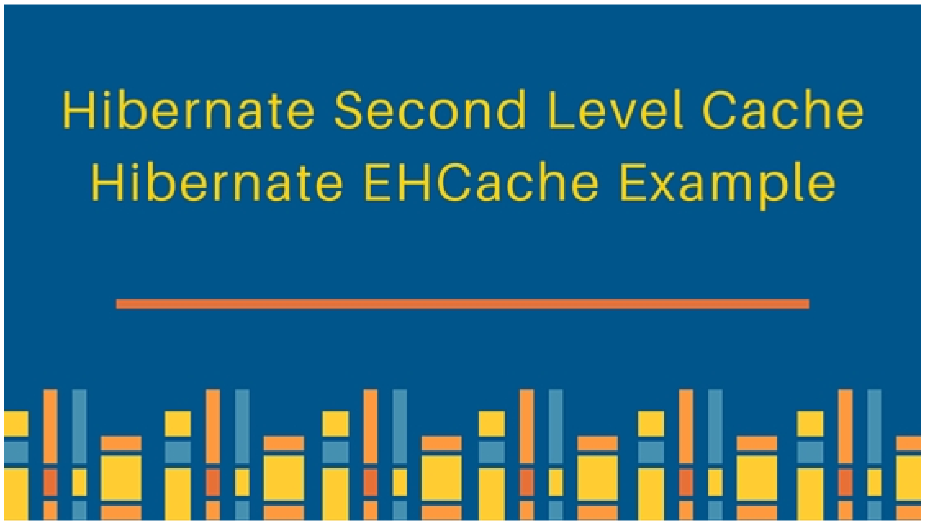
Hibernate EHCache – Hibernate Second Level Cache

APRIL 2, 2018 BY PANKAJ — 30 COMMENTS

Welcome to the Hibernate Second Level Cache Example Tutorial. Today we will look into Hibernate EHCache that is the most popular Hibernate Second Level Cache provider.

Table of Contents [hide]

Hibernate Second Level Cache



One of the major benefit of using Hibernate in large application is it's support for cache, hence reducing database queries and better performance. In earlier example, we looked into the Hibernate First Level Cache and today we will look into Hibernate Second Level Cache using **Hibernate EHCache** implementation.

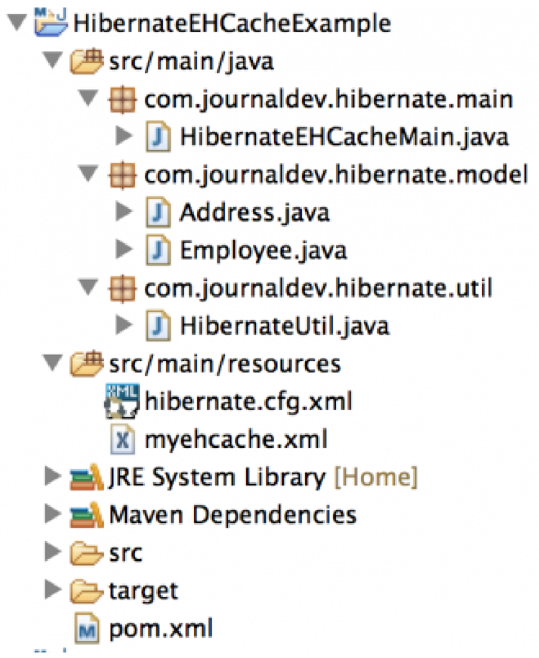
Hibernate Second Level cache providers include EHCache and Infinispan, but EHCache is more popular and we will use it for our example project. However before we move to our project, we should know different strategies for caching an object.

1. **Read Only**: This caching strategy should be used for persistent objects that will always read but never updated. It's good for reading and caching application configuration and other static data that are never updated. This is the simplest strategy with best performance because there is no overload to check if the object is updated in database or not.
2. **Read Write**: It's good for persistent objects that can be updated by the hibernate application. However if the data is updated either through backend or other applications, then there is no way hibernate will know about it and data might be stale. So while using this strategy, make sure you are using Hibernate API for updating the data.
3. **Nonrestricted Read Write**: If the application only occasionally needs to update data and strict transaction isolation is not required, a nonstrict-read-write cache might be appropriate.
4. **Transactional**: The transactional cache strategy provides support for fully transactional cache providers such as JBoss TreeCache. Such a cache can only be used in a JTA environment and you must specify hibernate.transaction.manager_lookup_class.

Hibernate EHCache

Since EHCache supports all the above cache strategies, it's the best choice when you are looking for second level cache in hibernate. I would not go into much detail about EHCache, my main focus will be to get it working for hibernate application.

Create a maven project in the Eclipse or your favorite IDE, final implementation will look like below image.


```
HibernateEHCacheExample
  src/main/java
    com.journaldev.hibernate.main
      HibernateEHCacheMain.java
    com.journaldev.hibernate.model
      Address.java
      Employee.java
    com.journaldev.hibernate.util
      HibernateUtil.java
  src/main/resources
    hibernate.cfg.xml
    myehcache.xml
  JRE System Library [Home]
  Maven Dependencies
  src
  target
  pom.xml
```

Let's look into each component of the application one by one.

Hibernate EHCache Maven Dependencies

For hibernate second level cache, we would need to add **ehcache-core** and **hibernate-ehcache** dependencies in our application. EHCache uses slf4j for logging, so I have also added **slf4j-simple** for logging purposes. I am using the latest versions of all these APIs, there is a slight chance that hibernate-ehcache APIs are not compatible with the ehcache-core API, in that case you need to check the pom.xml of hibernate-ehcache to find out the correct version to use. Our final pom.xml looks like below.

```
<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.journaldev.hibernate</groupId>
        <artifactId>HibernateEHCacheExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <description>Hibernate Secondary Level Cache Example using EHCache
implementation</description>

        <dependencies>
                <!-- Hibernate Core API -->
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>4.3.5.Final</version>
                </dependency>
                <!-- MySQL Driver -->
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
```

Hibernate Second Level Cache – Hibernate EHCache Configuration

Hibernate Second level cache is disabled by default, so we would need to enable it and add some configurations to get it working. Our hibernate.cfg.xml file looks like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "classpath://org/hibernate/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
                <property name="hibernate.connection.password">pankaj123</property>
                <property
name="hibernate.connection.url">jdbc:mysql://localhost/TestDB</property>
                <property name="hibernate.connection.username">pankaj</property>
                <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

                <property
name="hibernate.current_session_context_class">thread</property>
                <property name="hibernate.show_sql">true</property>

                <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFa
```

Some important points about hibernate second level cache configurations are:

1. **hibernate.cache.region.factory_class** is used to define the Factory class for Second level caching, I am using `org.hibernate.cache.ehcache.EhCacheRegionFactory` for this. If you want the factory class to be singleton, you should use `org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory` class.
   If you are using Hibernate 3, corresponding classes will be `net.sf.ehcache.hibernate.EhCacheRegionFactory` and `net.sf.ehcache.hibernate.SingletonEhCacheRegionFactory`.

2. **hibernate.cache.use_second_level_cache** is used to enable the second level cache.
3. **hibernate.cache.use_query_cache** is used to enable the query cache, without it HQL queries results will not be cached.

4. **net.sf.ehcache.configurationResourceName** is used to define the EHCache configuration file location, it's an optional parameter and if it's not present EHCache will try to locate ehcache.xml file in the application classpath.

Hibernate EHCache Configuration File

Our EHCache configuration file myehcache.xml looks like below.

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

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

        <defaultCache maxEntriesLocalHeap="10000" eternal="false"
                timeToIdleSeconds="120" timeToLiveSeconds="120"
diskSpoolBufferSizeMB="30"
                maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"
                memoryStoreEvictionPolicy="LRU" statistics="true">
                <persistence strategy="localTempSwap" />
        </defaultCache>

        <cache name="employee" maxEntriesLocalHeap="10000" eternal="false"
                timeToIdleSeconds="5" timeToLiveSeconds="10">
                <persistence strategy="localTempSwap" />
        </cache>

        <cache name="org.hibernate.cache.internal.StandardQueryCache"
                maxEntriesLocalHeap="5" eternal="false" timeToLiveSeconds="120">
```

Hibernate EHCache provides a lot of options, I won't go into much detail but some of the important configurations above are:

1. **diskStore**: EHCache stores data into memory but when it starts overflowing, it start writing data into file system. We use this property to define the location where EHCache will write the overflown data.
2. **defaultCache**: It's a mandatory configuration, it is used when an Object need to be cached and there are no caching regions defined for that.
3. **cache name="employee"**: We use cache element to define the region and it's configurations. We can define multiple regions and their properties, while defining model beans cache properties, we can also define region with caching strategies. The cache properties are easy to understand and clear with the name.
4. Cache regions `org.hibernate.cache.internal.StandardQueryCache` and `org.hibernate.cache.spi.UpdateTimestampsCache` are defined because EHCache was giving warning to that.

Hibernate Second Level Cache – Model Bean Caching Strategy

We use `org.hibernate.annotations.Cache` annotation to provide the caching configuration. `org.hibernate.annotations.CacheConcurrencyStrategy` is used to define the caching strategy and we can also define the cache region to use for the model beans.

```
package com.journaldev.hibernate.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name = "ADDRESS")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="employee")
public class Address {

        @Id
        @Column(name = "emp_id", unique = true, nullable = false)
```

```
package com.journaldev.hibernate.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
import org.hibernate.annotations.Cascade;

@Entity
```

```
@Table(name = "EMPLOYEE")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="employee")
public class Employee {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
```

Note that I am using the same database setup as in HQL example, you might want to check that to create the database tables and load sample data.

Hibernate SessionFactory Utility Class

We have a simple utility class to configure hibernate and get the `SessionFactory` singleton instance.

```
package com.journaldev.hibernate.util;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {

        private static SessionFactory sessionFactory;

        private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
                Configuration configuration = new Configuration();
                configuration.configure("hibernate.cfg.xml");
                System.out.println("Hibernate Configuration loaded");

                ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();
```

Our hibernate second level cache project using Hibernate EHCache is ready, let's write a simple program to test it.

Hibernate EHCache Test Program

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.stat.Statistics;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateEHCacheMain {

        public static void main(String[] args) {

                System.out.println("Temp Dir:"+System.getProperty("java.io.tmpdir"));

                //Initialize Sessions
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                Statistics stats = sessionFactory.getStatistics();
                System.out.println("Stats enabled="+stats.isStatisticsEnabled());
                stats.setStatisticsEnabled(true);
                System.out.println("Stats enabled="+stats.isStatisticsEnabled());
```

`org.hibernate.stat.Statistics` provides the statistics of Hibernate SessionFactory, we are using it to print the fetch count and second level cache hit, miss and put count. Statistics are disabled by default for better performance, that's why I am enabling it at the start of the program.

When we run above program, we get a lot of output generated by Hibernate and EHCache APIs, but we are interested in the data that we are printing. A sample run prints following output.

```
Temp Dir:/var/folders/h4/q73jjy0902g51wkw0w69c0600000gn/T/
Hibernate Configuration loaded
Hibernate serviceRegistry created
Stats enabled=false
Stats enabled=true
***** 0 *****
Fetch Count=0
Second Level Hit Count=0
Second Level Miss Count=0
Second Level Put Count=0
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer
join ADDRESS address1_ on employee0_.emp_id=address1_.emp_id where
employee0_.emp_id=?
1:: Name=Pankaj, Zipcode=95129
***** 1 *****
```

```
Fetch Count=1
Second Level Hit Count=0
Second Level Miss Count=1
Second Level Put Count=2
```

As you can see from output, statistics were disabled at first but we enabled it for checking our hibernate second level cache.

Step by step explanation of the output is as follows:

1. Before we load any data in our application, all the stats are 0 as expected.
2. When we are loading the Employee with id=1 for the first time, it's first searched into first level cache and then second level cache. If not found in cache, database query is executed and hence fetch count becomes 1. Once the object is loaded, it's saved into first level cache and second level cache both. So secondary level hit count remains 0 and miss count becomes 1. Notice that put count is 2, that is because Employee object consists of Address too, so both the objects are saved into second level cache and count is increased to 2.
3. Next, we are again loading the employee with id=1, this time it's present in the first level cache. So you don't see any database query and all other secondary level cache stats also remains same.
4. Next we are using `evict()` method to remove the employee object from the first level cache, now when we are trying to load it, hibernate finds it in the second level cache. That's why no database query is fired and fetch count remains 1. Notice that hit count goes from 0 to 2 because both Employee and Address objects are read from the second level cache. Second level miss and put count remains at the earlier value.
5. Next we are loading an employee with id=3, database query is executed and fetch count increases to 2, miss count increases from 1 to 2 and put count increases from 2 to 4.
6. Next we are trying to load employee with id=1 in another session, Since hibernate second level cache is shared across sessions, it's found in the second level cache and no database query is executed. Fetch count, miss count and put count remains same whereas hit count increases from 2 to 4.

So it's clear that our Hibernate second level cache; Hibernate EHCache; is working fine. Hibernate statistics are helpful in finding the bottleneck in the system and optimize it to reduce the fetch count and load more data from the cache.

That's all for the **Hibernate EHCache example**, I hope it will help you in configuring EHCache in your hibernate applications and getting better performance through hibernate second level cache. You can download the sample project from below link and use other stats data to learn more.

Download Hibernate EHCache Project

« PREVIOUS
Hibernate Caching – First Level Cache

NEXT »
Hibernate Log4j Logging

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: HIBERNATE

Comments

Ning says
MAY 31, 2018 AT 2:52 PM
you should set the value of timeToLiveSeconds a littler bigger.
Reply

Arun SIngh says
OCTOBER 26, 2017 AT 12:33 PM
this is very helpful
Reply

Shashank says
MAY 15, 2017 AT 12:12 PM
too good. Worked for me. Great explanation.
Reply

nithin says
JANUARY 10, 2017 AT 2:06 AM
i am using criteria query or HQL to retrieve a list of values and i am not using query level cache.will second level cache work for it as each time i am trying its hitting the database
Reply

Ajay says
JANUARY 3, 2017 AT 4:42 AM
Hi,
When I am executing this project I am getting the following error:

```
Initial SessionFactory creation failed.java.lang.NoSuchMethodError:
javax.persistence.OneToOne.orphanRemoval()Z
java.lang.NoSuchMethodError: javax.persistence.OneToOne.orphanRemoval()Z
at org.hibernate.cfg.AnnotationBinder.processElementAnnotations(AnnotationBinder.java:1836)
at org.hibernate.cfg.AnnotationBinder.processIdPropertiesIfNotAlready(AnnotationBinder.java:963)
at org.hibernate.cfg.AnnotationBinder.bindClass(AnnotationBinder.java:796)
at
org.hibernate.cfg.Configuration$MetadataSourceQueue.processAnnotatedClassesQueue(Configuration.java:3788)
at org.hibernate.cfg.Configuration$MetadataSourceQueue.processMetadata(Configuration.java:3742)
at org.hibernate.cfg.Configuration.secondPassCompile(Configuration.java:1410)
at org.hibernate.cfg.Configuration.buildSessionFactory(Configuration.java:1844)
at com.journaldev.hibernate.util.HibernateUtil.buildSessionFactory(HibernateUtil.java:22)
at com.journaldev.hibernate.util.HibernateUtil.getSessionFactory(HibernateUtil.java:34)
at com.journaldev.hibernate.main.HibernateEHCacheMain.main(HibernateEHCacheMain.java:18)
Exception in thread "main" java.lang.ExceptionInInitializerError
at com.journaldev.hibernate.util.HibernateUtil.buildSessionFactory(HibernateUtil.java:29)
at com.journaldev.hibernate.util.HibernateUtil.getSessionFactory(HibernateUtil.java:34)
at com.journaldev.hibernate.main.HibernateEHCacheMain.main(HibernateEHCacheMain.java:18)
Caused by: java.lang.NoSuchMethodError: javax.persistence.OneToOne.orphanRemoval()Z
at org.hibernate.cfg.AnnotationBinder.processElementAnnotations(AnnotationBinder.java:1836)
at org.hibernate.cfg.AnnotationBinder.processIdPropertiesIfNotAlready(AnnotationBinder.java:963)
at org.hibernate.cfg.AnnotationBinder.bindClass(AnnotationBinder.java:796)
at
org.hibernate.cfg.Configuration$MetadataSourceQueue.processAnnotatedClassesQueue(Configuration.java:3788)
at org.hibernate.cfg.Configuration$MetadataSourceQueue.processMetadata(Configuration.java:3742)
at org.hibernate.cfg.Configuration.secondPassCompile(Configuration.java:1410)
at org.hibernate.cfg.Configuration.buildSessionFactory(Configuration.java:1844)
at com.journaldev.hibernate.util.HibernateUtil.buildSessionFactory(HibernateUtil.java:22)
… 2 more
```

Reply

Shambhu says
JULY 7, 2016 AT 12:51 AM
I modified the CacheConcurrencyStrategy.READ_ONLY to CacheConcurrencyStrategy.READ_WRITE.
But still it is not allowing to update the record.
I am getting below error
Exception in thread "main" java.lang.UnsupportedOperationException: Can't write to a readonly object
Reply

anonymous says
FEBRUARY 16, 2016 AT 11:39 AM
I wanted to implement caching to an existing project. In the project I can see pom.xml and persistence.xml but no hibernate.cfg.xml in the project. Could you assist on this as I am fairly new to caching and spring and hibernate. Thanks!
Reply

RV says
JANUARY 26, 2016 AT 5:31 AM
very helpful article. I have a doubt. How we can update/flush the contents of second level cache.
Suppose if a particular cached entry got updated.
Reply

kartik says
JUNE 15, 2016 AT 2:23 AM

there is not way u can update , although u can put to Time to live param in ehache cache and if query result doesnt find in cache , if will be loaded.
Reply

bablu says
SEPTEMBER 14, 2015 AT 6:12 AM
how we can Implement second level cache with spring Configuration
Reply

Ajit Dandapat says
AUGUST 5, 2015 AT 12:00 AM
Hi Pankaj,
Nice explanation regarding ehcache.
Did you remember me. TCS office Apple project (GR Techpark).
Reply

vnp says
APRIL 9, 2015 AT 2:40 AM
Thanks for this tutorial.
By following the tutorial i got cache working for criteria, but it is not working for HQL (though "use_query_cache" is set)
Reply

Arvind Ojha says
APRIL 4, 2015 AT 2:18 AM
It is nice explanation about second level cache …
Reply

Chintan says
MARCH 19, 2015 AT 12:34 AM
Hi Pankaj,
This is really wonderful tutorial for 2nd level caching. It is now pretty clear for me. However when I tried to copy the code and execute it is showing different output:

```
Stats Enable: false
Stats Enable: true
********** 0 **********
Fetch Count: 0
Second level hit count: 0
Second level miss count: 0
Second level put count: 0
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.salary as salary3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.emp_address_line1 as emp_address_line2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from employee2 employee0_ left outer join address
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
1 :: Name: Pankaj, Zipcode: 95129
********** 1 **********
Fetch Count: 1
Second level hit count: 0
Second level miss count: 0
Second level put count: 1
2 :: Name: Pankaj, Zipcode: 95129
********** 2 **********
Fetch Count: 1
Second level hit count: 0
Second level miss count: 0
Second level put count: 1
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.salary as salary3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.emp_address_line1 as emp_address_line2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from employee2 employee0_ left outer join address
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
3 :: Name: Pankaj, Zipcode: 95129
********** 3 **********
Fetch Count: 2
Second level hit count: 0
Second level miss count: 0
Second level put count: 1
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.salary as salary3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.emp_address_line1 as emp_address_line2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from employee2 employee0_ left outer join address
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
4 :: Name: Lisa, Zipcode: 560100
********** 4 **********
Fetch Count: 3
Second level hit count: 0
Second level miss count: 0
Second level put count: 2
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.salary as salary3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.emp_address_line1 as emp_address_line2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from employee2 employee0_ left outer join address
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
5 :: Name: Pankaj, Zipcode: 95129
********** 5 **********
Fetch Count: 4
Second level hit count: 0
Second level miss count: 0
Second level put count: 2
```

Can you please explain me why it is.

Note: I am using Oracle 11g database. All other things are same as this tutorial.

Reply

Ning says

MAY 31, 2018 AT 2:54 PM

you can set the timeToLiveSeconds and timeToIdleSeconds bigger.

Reply

Sriram says

NOVEMBER 25, 2014 AT 10:12 AM

Hi Pankaj,

That is really a great piece of explanation and your code just works without any glitch.

just take the code, put it in the IDE, set up the DB and I was good to go.

Wonderful piece and just keep that up!

Regards,

Sriram

Reply

Siva says

OCTOBER 11, 2014 AT 8:03 PM

what if data is updated which is already been cached?

Eg: There is an update operation on empid -1. Then what happens?

Reply

vnp says

APRIL 9, 2015 AT 2:42 AM
you should use apt querying technique
Reply

Seetesh says
OCTOBER 7, 2014 AT 5:55 AM
failed.org.hibernate.service.spi.ServiceException: Unable to create requested service [org.hibernate.cache.spi.RegionFactory]
Any clues on what is going wrong?
Reply

Sriram says
NOVEMBER 25, 2014 AT 10:14 AM
Hi Seetesh,
You may have to post the log more in detail so that we could analyse.
Thanks!
Sriram
Reply

hanan mahmoud says
SEPTEMBER 29, 2014 AT 3:53 AM
So clear,easy,straight forward example.and the test class really clarify the concept of 2nd level caching
Reply

Abhinav says
SEPTEMBER 9, 2014 AT 10:57 PM
what is region? how it works and how to define region through annotation?
Reply

Abhinav says
SEPTEMBER 9, 2014 AT 10:55 PM
What is region here:
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="employee")
Is it referring to :

Why are we using that? if we dont use, then what will happen? and how to define that 'region' through annotation?
Thanks.
Reply

sankar says
SEPTEMBER 6, 2014 AT 6:25 PM
HI Pankaj,
I'm trying to configure clustered cache with hibernate. I'm able to run the program successfully, but I'm not able to see the cache in Terracotta(EHCache).
could you please let me know if you have any thoughts on this?
Reply

ravinder says
AUGUST 29, 2014 AT 4:22 AM
Hello bro!
I am ravinder, working as a software engineer But I am beginner for webServices and Hibernate. So I would like to request to how can configure the webservices and hibernate with eclipse ide.So further what is required things can you send through "URL'S(like jar and war file) " and along with the "screen shorts ".If you send with example also it's great.
Advance,
Thanks and regards
ravinder
Reply

Vikas Gandham says
AUGUST 27, 2014 AT 1:14 PM
I have set up 2nd level cache in our application.For the first call of session.get for any object it fetches from the database and for the 2nd call it gets from cache(found out from postgres logs).The issue is if the object has boolean attributes these attributes has null when fetched from cache though its value is true/false.Does any one faced this issue please let me know. Strucked with this for almost 4 days and did not find the solution
Reply

Avnish says
AUGUST 27, 2014 AT 2:13 AM
Thanks Pankaj for a very helpful doc and to understand EHCache.

Reply

Ahmet says

AUGUST 13, 2014 AT 5:15 AM

Thank you so much. God bless you.

Reply

sujeet says

JUNE 20, 2014 AT 3:49 AM

Thanks Pankaj for detailed explanation

Reply

Pankaj says

JUNE 29, 2014 AT 6:12 AM

you are welcome sujeet.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

HIBERNATE FRAMEWORK

Hibernate Caching

- Hibernate Cache
- Hibernate EHCache

Hibernate Integrations

- Hibernate Spring
- Hibernate Spring MVC
- Hibernate Struts 2
- Hibernate Primefaces
- Hibernate Primefaces Spring
- Hibernate SpringRoo Primefaces
- Hibernate JSF Spring

Miscellaneous

- Hibernate Tools Eclipse Plugin
- Hibernate Configuration Offline
- [Solved] No identifier specified
- Hibernate Program Not Terminating
- Access to DialectResolutionInfo
- get is not valid
- No CurrentSessionContext configured
- Hibernate Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial