Hibernate EHCache Second

Pankaj June 4, 2014 Hibernate

One of the major benefit of using Hibernate in large application is it's support for caching, 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

ıls: Java IO Tutorial, Expressions Multi-Threading Logging API Tutorial, Java Annotations Tutorial,Java XML Tutorial, Java Collections Tutorial, Java Generics Tutorial, Java Exception Handling, Java Reflection Tutorial, Java Design Patterns, JDBC Tutorial
Java EE: Servlet JSP Tutorial, Struts2 Tutorial, Spring Tutorial, Hibernate Tutorial, Primefaces Tutorial
Web Services: Apache Axis 2 Tutorial, Jersey Restful Web Services Tutorial
Misc: Memcached Tutorial

Resources: Free eBooks, My Favorite Web Hosting

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.

Since EHCache supports all the above caching strategies, it's the best choice when you are looking for second level caching 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.

Interview Questions

Java String Interview Questions, Java Multi-Threading Interview Questions, Java Programming Interview Questions, Java Interview Questions, Java Collections Interview Questions, Java Exception Interview Questions

Servlet Interview Questions, JSP Interview Questions, Struts2 Interview Questions, JDBC Interview Questions, Spring Interview Questions, Hibernate Interview Questions

Pages

About

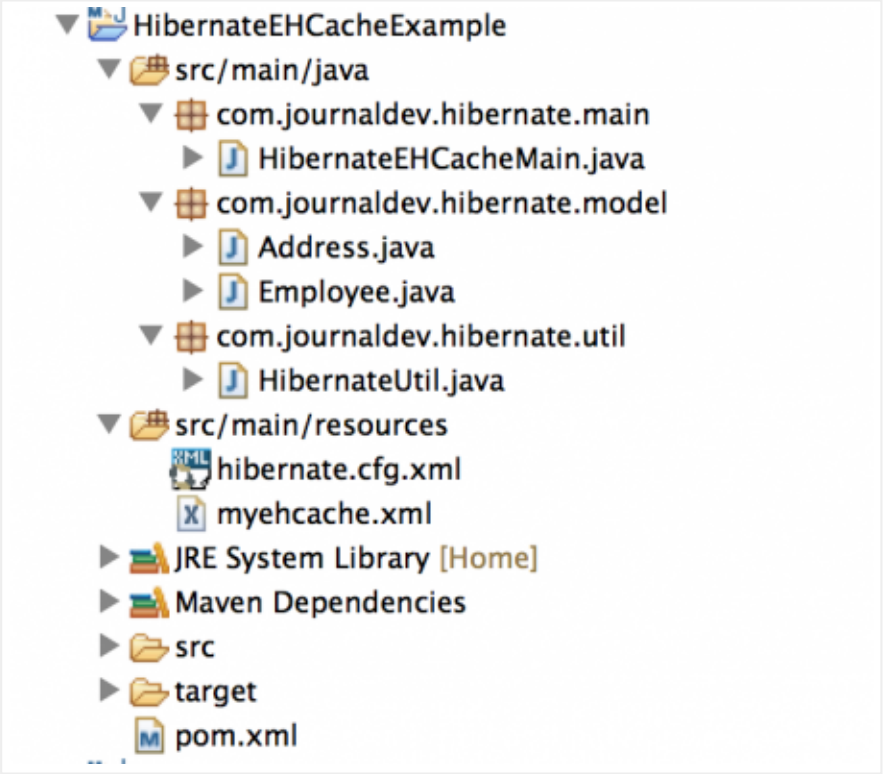
Advertise

Java Interview Questions

PII Privacy Policy

Privacy Policy

Resources



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.

pom.xml

```
<project xmlns="http://maven.apache.org/POM
    xsi:schemaLocation="http://maven.apache
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.journaldev.hibernate</grou
    <artifactId>HibernateEHCacheExample</ar
    <version>0.0.1-SNAPSHOT</version>
    <description>Hibernate Secondary Level

    <dependencies>
        <!-- Hibernate Core API -->
        <dependency>
            <groupId>org.hibernate</groupId
```

```
            <artifactId>hibernate-core</art
            <version>4.3.5.Final</version>
        </dependency>
        <!-- MySQL Driver -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-jav
            <version>5.0.5</version>
        </dependency>
        <!-- EHCache Core APIs -->
        <dependency>
            <groupId>net.sf.ehcache</groupI
            <artifactId>ehcache-core</artif
            <version>2.6.9</version>
        </dependency>
        <!-- Hibernate EHCache API -->
        <dependency>
            <groupId>org.hibernate</groupId
            <artifactId>hibernate-ehcache</
            <version>4.3.5.Final</version>
        </dependency>
        <!-- EHCache uses slf4j for logging
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artif
            <version>1.7.5</version>
        </dependency>
    </dependencies>
</project>
```

Hibernate Configuration for Second Level EHCache

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

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "c
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connectio
        <property name="hibernate.connectio
        <property name="hibernate.connectio
        <property name="hibernate.connectio
        <property name="hibernate.dialect">

        <property name="hibernate.current_s
        <property name="hibernate.show_sql"

        <property name="hibernate.cache.reg
```

```
15
16          <!-- For singleton factory -->
17          <!-- <property name="hibernate.cach
18           -->
19
20           <!-- enable second level cache and
21           <property name="hibernate.cache.us
22           <property name="hibernate.cache.us
23           <property name="net.sf.ehcache.con
24
25          <mapping class="com.journaldev.hibe
26          <mapping class="com.journaldev.hibe
27      </session-factory>
28  </hibernate-configuration>
```

Some important points about hibernate 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.

EHCache Configuration File

Our EHCache configuration file looks like below.

myehcache.xml

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

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

    <defaultCache maxEntriesLocalHeap="1000
        timeToIdleSeconds="120" timeToLiveS
        maxEntriesLocalDisk="10000000" disk
        memoryStoreEvictionPolicy="LRU" sta
        <persistence strategy="localTempSwa
    </defaultCache>

    <cache name="employee" maxEntriesLocalH
        timeToIdleSeconds="5" timeToLiveSec
        <persistence strategy="localTempSwa
    </cache>

    <cache name="org.hibernate.cache.intern
        maxEntriesLocalHeap="5" eternal="fa
        <persistence strategy="localTempSwa
    </cache>

    <cache name="org.hibernate.cache.spi.Up
        maxEntriesLocalHeap="5000" eternal=
        <persistence strategy="localTempSwa
    </cache>
</ehcache>
```

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 Model Beans with 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.

Address.java

```
1  package com.journaldev.hibernate.model;
2
3  import javax.persistence.Column;
4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.Id;
7  import javax.persistence.OneToOne;
8  import javax.persistence.PrimaryKeyJoinColu
9  import javax.persistence.Table;
10
11 import org.hibernate.annotations.Cache;
12 import org.hibernate.annotations.CacheConcu
13 import org.hibernate.annotations.GenericGen
14 import org.hibernate.annotations.Parameter;
15
16 @Entity
17 @Table(name = "ADDRESS")
18 @Cache(usage=CacheConcurrencyStrategy.READ_
19 public class Address {
20
21     @Id
22     @Column(name = "emp_id", unique = true,
23     @GeneratedValue(generator = "gen")
24     @GenericGenerator(name = "gen", strateg
25                 parameters = { @Parameter(n
26     private long id;
27
28     @Column(name = "address_line1")
29     private String addressLine1;
30
31     @Column(name = "zipcode")
32     private String zipcode;
33
34     @Column(name = "city")
35     private String city;
36
37     @OneToOne
```

```
38      @PrimaryKeyJoinColumn
39      private Employee employee;
40
41      public long getId() {
42          return id;
43      }
44
45      public void setId(long id) {
46          this.id = id;
47      }
48
49      public String getAddressLine1() {
50          return addressLine1;
51      }
52
53      public void setAddressLine1(String addr
54          this.addressLine1 = addressLine1;
55      }
56
57      public String getZipcode() {
58          return zipcode;
59      }
60
61      public void setZipcode(String zipcode)
62          this.zipcode = zipcode;
63      }
64
65      public String getCity() {
66          return city;
67      }
68
69      public void setCity(String city) {
70          this.city = city;
71      }
72
73      public Employee getEmployee() {
74          return employee;
75      }
76
77      public void setEmployee(Employee employ
78          this.employee = employee;
79      }
80
81  }
```

Employee.java

```
1   package com.journaldev.hibernate.model;
2
3   import javax.persistence.Column;
4   import javax.persistence.Entity;
5   import javax.persistence.GeneratedValue;
6   import javax.persistence.GenerationType;
7   import javax.persistence.Id;
8   import javax.persistence.OneToOne;
9   import javax.persistence.Table;
10
11  import org.hibernate.annotations.Cache;
12  import org.hibernate.annotations.CacheConcu
13  import org.hibernate.annotations.Cascade;
```

```
14
15  @Entity
16  @Table(name = "EMPLOYEE")
17  @Cache(usage=CacheConcurrencyStrategy.READ_
18  public class Employee {
19
20      @Id
21      @GeneratedValue(strategy = GenerationTy
22      @Column(name = "emp_id")
23      private long id;
24
25      @Column(name = "emp_name")
26      private String name;
27
28      @Column(name = "emp_salary")
29      private double salary;
30
31      @OneToOne(mappedBy = "employee")
32      @Cascade(value = org.hibernate.annotati
33      private Address address;
34
35      public long getId() {
36          return id;
37      }
38
39      public void setId(long id) {
40          this.id = id;
41      }
42
43      public Address getAddress() {
44          return address;
45      }
46
47      public void setAddress(Address address)
48          this.address = address;
49      }
50
51      public String getName() {
52          return name;
53      }
54
55      public void setName(String name) {
56          this.name = name;
57      }
58
59      public double getSalary() {
60          return salary;
61      }
62
63      public void setSalary(double salary) {
64          this.salary = salary;
65      }
66
67  }
```

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.

```
il.java
e com.journaldev.hibernate.util;

 org.hibernate.SessionFactory;
 org.hibernate.boot.registry.StandardServiceRegistr
 org.hibernate.cfg.Configuration;
 org.hibernate.service.ServiceRegistry;

 class HibernateUtil {

ivate static SessionFactory sessionFactory;

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

      ServiceRegistry serviceRegistry = new Standa
      System.out.println("Hibernate serviceRegistr

      SessionFactory sessionFactory = configuratio

      return sessionFactory;
  }
  catch (Throwable ex) {
      System.err.println("Initial SessionFactory c
      ex.printStackTrace();
      throw new ExceptionInInitializerError(ex);
  }


blic static SessionFactory getSessionFactory() {
  if(sessionFactory == null) sessionFactory = buil
  return sessionFactory;
```

Our hibernate second level EHCache setup is ready, let's write a simple program to test it.

Hibernate EHCache Test Program

HibernateEHCacheMain.java

```
1  package com.journaldev.hibernate.main;
2
```

```
3  import org.hibernate.Session;
4  import org.hibernate.SessionFactory;
5  import org.hibernate.Transaction;
6  import org.hibernate.stat.Statistics;
7
8  import com.journaldev.hibernate.model.Emplo
9  import com.journaldev.hibernate.util.Hibern
10
11 public class HibernateEHCacheMain {
12
13     public static void main(String[] args)
14
15         System.out.println("Temp Dir:"+Syst
16
17         //Initialize Sessions
18         SessionFactory sessionFactory = Hib
19         Statistics stats = sessionFactory.g
20         System.out.println("Stats enabled="
21         stats.setStatisticsEnabled(true);
22         System.out.println("Stats enabled="
23
24         Session session = sessionFactory.op
25         Session otherSession = sessionFacto
26         Transaction transaction = session.b
27         Transaction otherTransaction = othe
28
29         printStats(stats, 0);
30
31         Employee emp = (Employee) session.l
32         printData(emp, stats, 1);
33
34         emp = (Employee) session.load(Emplo
35         printData(emp, stats, 2);
36
37         //clear first level cache, so that
38         session.evict(emp);
39         emp = (Employee) session.load(Emplo
40         printData(emp, stats, 3);
41
42         emp = (Employee) session.load(Emplo
43         printData(emp, stats, 4);
44
45         emp = (Employee) otherSession.load(
46         printData(emp, stats, 5);
47
48         //Release resources
49         transaction.commit();
50         otherTransaction.commit();
51         sessionFactory.close();
52     }
53
54     private static void printStats(Statisti
55         System.out.println("***** " + i + "
56         System.out.println("Fetch Count="
57                 + stats.getEntityFetchCount
58         System.out.println("Second Level Hi
59                 + stats.getSecondLevelCache
60         System.out
61                 .println("Second Level Miss
62                         + stats
63                                 .getSecondL
```

```
64          System.out.println("Second Level Pu
65                  + stats.getSecondLevelCache
66      }
67
68      private static void printData(Employee
69          System.out.println(count+":: Name="
70          printStats(stats, count);
71      }
72
73  }
```

`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.

```
1   Temp Dir:/var/folders/h4/q73jjy0902g51wkw0w
2   Hibernate Configuration loaded
3   Hibernate serviceRegistry created
4   Stats enabled=false
5   Stats enabled=true
6   ***** 0 *****
7   Fetch Count=0
8   Second Level Hit Count=0
9   Second Level Miss Count=0
10  Second Level Put Count=0
11  Hibernate: select employee0_.emp_id as emp_
12  1:: Name=Pankaj, Zipcode=95129
13  ***** 1 *****
14  Fetch Count=1
15  Second Level Hit Count=0
16  Second Level Miss Count=1
17  Second Level Put Count=2
18  2:: Name=Pankaj, Zipcode=95129
19  ***** 2 *****
20  Fetch Count=1
21  Second Level Hit Count=0
22  Second Level Miss Count=1
23  Second Level Put Count=2
24  3:: Name=Pankaj, Zipcode=95129
25  ***** 3 *****
26  Fetch Count=1
27  Second Level Hit Count=2
28  Second Level Miss Count=1
29  Second Level Put Count=2
30  Hibernate: select employee0_.emp_id as emp_
31  4:: Name=Lisa, Zipcode=560100
32  ***** 4 *****
```

```
Fetch Count=2
Second Level Hit Count=2
Second Level Miss Count=2
Second Level Put Count=4
5:: Name=Pankaj, Zipcode=95129
***** 5 *****
Fetch Count=2
Second Level Hit Count=4
Second Level Miss Count=2
Second Level Put Count=4
```

As you can see from output, statistics were disabled at first but we enabled it for checking our 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 out EHCache second level hibernate cache is working fine and as expected. 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 gaining better performance. You can download the sample project from below link and use other stats data to learn more.

Hibernate EHCache Project
1893 downloads

Related Posts:

hibernate | hibernate cache | hibernate ehcache | hibernate example | hibernate tutorial

Written by 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**.

20 Responses to "Hibernate EHCache Second Level Caching Example Tutorial"

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

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 *

Sign me up for the JournalDev newsletter!

Website

Post Comment