Using Second Level Caching in a JPA Application

- January 3, 2014
- By Manoj Debnath
- Bio »
- Send Email »
- More Articles »

Caching structurally implies a temporary store to keep data for quicker access later on. Second level shared cache is an auxiliary technique, mainly used in JPA, to enhance performance; it is used especially during large inflow, outflow of data between the database and the application. Caching also reduces search time when the searched entity is already in the cache, otherwise it is fetched from the database to serve the purpose. However, when a subsequent search query is fired it takes little time as the searched entity is already in the cache. Entity Manager stores entities for almost every CRUD operation in this shared cache before flushing the content to the database. Second level cache is complementary to its first level counterpart. In fact, to get an overview of caching in particular, there are several layers of caching in JPA.

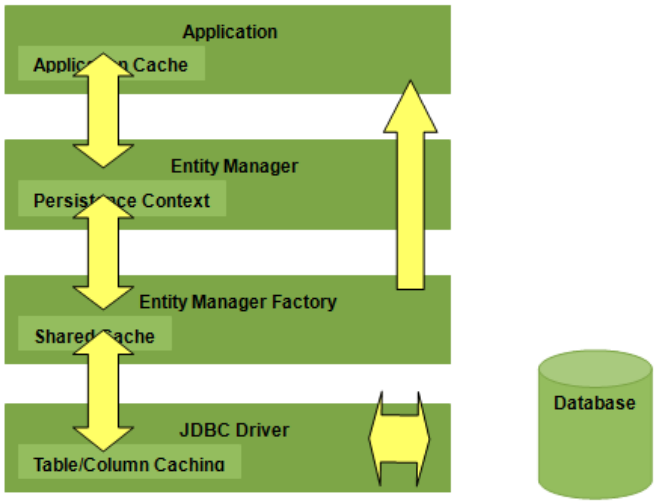


Figure 1: JPA layers of Caching

First Level Cache

Entity Manager always maintains a cache called the first level cache, so it makes sense to call another shared caching technique in addition to first – a second level cache. In first level cache CRUD operations are performed per transaction basis to reduce the number of queries fired to the database. That is, an entity modified several times within the same transaction is done in the cache only, modification at the database level is slated until final UPDATE statement is fired at the end of the transaction. As illustrated in the **figure 2**, JPA entities are cached at the persistence context level and guarantees that there will be one object instance per persistence context for a specific row of a database table. Concurrent transactions affecting the same row are managed by applying an appropriate locking mechanism in JPA.

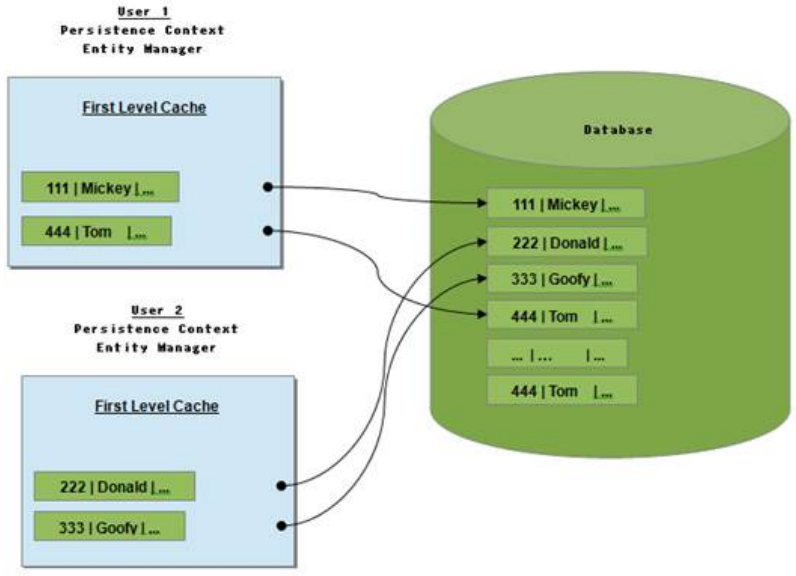


Figure 2: First level Cache

Second Level Cache

This level of cache emerged more due to performance reasons than absolute necessity; in fact adding more shared cache increases the possibility of the problem of a stale read. As illustrated in **figure 3**, second level cache sits between Entity Manager and the database. Persistence context shares the cache, making the second level cache available throughout the application. Database traffic is reduced considerably because entities are loaded in to the shared cache and made available from there. So in a nutshell second level cache provides the following benefits.

- Persistence provider manages local store of entity data
- Leverages performance by avoiding expensive database calls
- Data are kept transparent to the application
- CRUD operation can be performed through normal entity manager functions
- Application can remain oblivious of the underlying cache and do its job inadvertently

With Java EE 7, JPA 2.0 acknowledged the need of a second level cache and has provided very minimal APIs. Perhaps, it is still undecided whether JPA would fully support a functional second level cache in the near future or not. However, persistence providers like EclipseLink, Hibernate, etc. have their own set of implementation created either reusing an existing one or developing their own from scratch. As a result, a large part of the shared cache API is not standardized. The good news is JPA 2.0 nodded at last and we shall wait to see its future course in its specification.

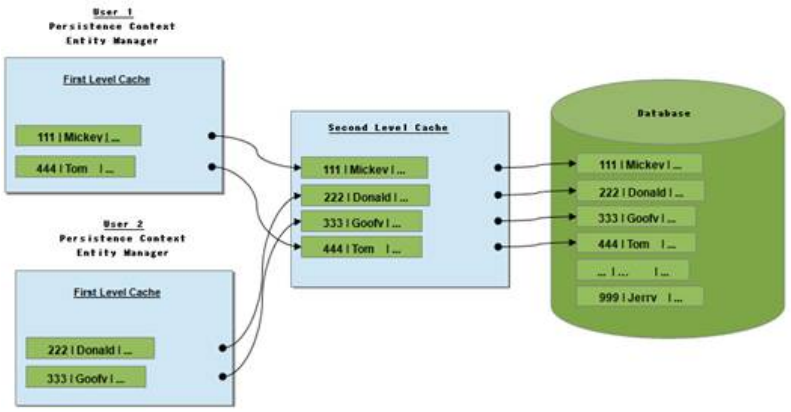


Figure 3: Second level Cache

Implementing Second Level Cache

Related Articles

- How to Create a JVM Instance in JNI
- Getting Started with Android Using Java
- Exception Handling in JNI
- Manipulating Java Objects in Native Code
- Desired Properties of a Big Data System
- The Java 7 Features Bound to Make Developers More Productive
- Jetty Continuations: Push Your Java Server Beyond Its Scalability Limits
- Java Multi-threading and the Challenges of Parallel Computing

The *javax.persistence.Cache* interface of the persistence provider can be used to interact with the second level cache. This interface provides functions such as, *contains*: to check whether the cache contains a given (as parameter) entity, variation of *evict*: to remove a particular entity from the cache, *evictAll*: to clear the cache and *unwrap*: to return specified cache implementation by the provider (ref. Javadoc JPA API). We can use @Cacheable annotation to make a POJO eligible to be cached. In this way the persistence provider knows which entity is to be cached. If no such annotation is supplied then entity and its state is not cached by the provider. @Cacheable takes a parameter of boolean value, default is true.

Listing 1: using @Cacheable annotation

```
package org.mano.dto;
@Entity
@Cacheable(true)
public class Account{
        @Id @GeneratedValue(strategy=GenerationType.AUTO)
        private Long accountNumber;
        private String name;
        @Temporal(TemporalType.DATE)
        private Date createDate;
        private Float balance;

        //... constructors, getters, setters
}
```

Once the entity is all set to be cache-able, we need to provide an appropriate caching mechanism to be used by the persistence provider in *persistence.xml*. This is done by setting one of the following values in ***shared-cached-mode*** element such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="...">
<persistence-unit name="JPALockingDemo" transaction-type="RESOURCE_LOCAL">
```

```
<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
<class>org.mano.dto.Account</class>

<shared-cache-mode>ALL</shared-cache-mode>

<properties>
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://Localhost:3306/bank"/>
<property name="javax.persistence.jdbc.user" value="bank"/>
<property name="javax.persistence.jdbc.password" value="secret"/>
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<property name="javax.persistence.schema-generation.database.action" value="create"/>
</properties>
</persistence-unit>
</persistence>
```

Value	Description
ALL	All entities are cached
NONE	Disable caching
ENABLE_SELECTIVE	Enable caching only for those entities which has @Cacheable(true)
DISABLE_SELECTIVE	Enable caching only for those entities which are not specified with @Cacheable(false)
UNSPECIFIED	Applies persistence provider-specific default behavior

The POJO *Account* in **Listing 1** is now cache-able, code in **Listing 2** will demonstrate the case.

Listing 2: Demonstrating the use of cache-able POJO

```
package org.mano.app;

public class Main {

	public static void main(String[] args) {
		EntityManagerFactory factory = Persistence
				.createEntityManagerFactory("JPALockingDemo");
		EntityManager manager = factory.createEntityManager();

		Account account = new Account(111l,"Patrick",new Date(),23000f);
		manager.getTransaction().begin();
		manager.persist(account);
		manager.getTransaction().commit();

		Cache cache = factory.getCache();
		System.out.println("cache.contain should return true: "+cache.contains(Account.class, account.getAccountNumber()));
		cache.evict(Account.class);
		System.out.println("cache.contain should return false: "+cache.contains(Account.class, account.getAccountNumber()));
		manager.close();
		factory.close();
	}
}
```

Conclusion

JPA caching is flexible enough to configure per class basis or globally with the help of persistence unit settings or class settings. We can either set ***shared-cache-mode*** element in the *persistence.xml* or dynamically set *javax.persistence.sharedCache.mode* property while creating entity manager factory. In concurrent transactions, setting the cache mode to NONE in view of better performance may on the contrary lead to performance slowdown and is not recommended; such situations should rather be handled with an appropriate locking mechanism.

Tags: Java, performance, application, JPA

3 Comments (click to add your comment)
By sentil June 09 2015 02:56 PDT

Hi your post is extremely useful to our team Regards sentil

Reply to this comment By prem May 18 2015 06:51 PDT

I understood lot from this caching, i would suggest to have a download option to get the code for testing this.

Reply to this commentBy tulsi hemrajani March 17 2015 01:02 PDT

I have created a POC to implement Second level cache. I have followed the steps as given in the article , but I m unable to reach out to the solution. Kinldy reply back so I can send you the details of my POC .

Reply to this comment

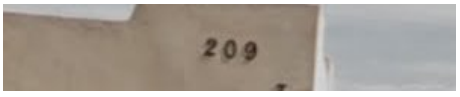
Comment and Contribute

Your name/nickname

Your email

Subject

(Maximum characters: 1200). You have 1200 characters left.



Type the text

Privacy & Terms

Submit Your Comment

Enterprise Development Update

Don't miss an article. Subscribe to our newsletter below.

Enter Email Address SIGN UP

Most Popular Developer Stories

- Today
- This Week
- All-Time

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 MIDP Programming with J2ME
- 4 An Introduction to JSP Standard Template Library (JSTL)
- 5 Debugging a Java Program with Eclipse

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 An Introduction to JSP Standard Template Library (JSTL)
- 4 MIDP Programming with J2ME
- 5 Debugging a Java Program with Eclipse

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 An Introduction to JSP Standard Template Library (JSTL)
- 4 MIDP Programming with J2ME
- 5 Debugging a Java Program with Eclipse

Most Commented On

- This Week
- This Month
- All-Time

- 1 10 Experimental PHP Projects Pushing the

Envelope
- 2 Day 1: Learning the Basics of PL/SQL
- 3 C# Tip: Placing Your C# Application in the System Tray
- 4 Logical Versus Physical Database Modeling
- 5 Is Ubuntu Contributing as Much as It Should to Free Software Projects?

- 1 Day 1: Learning the Basics of PL/SQL
- 2 The 5 Developer Certifications You'll Wish You Had in 2015
- 3 10 Experimental PHP Projects Pushing the Envelope
- 4 An Introduction to Struts
- 5 Inside Facebook's Open Source Infrastructure

- 1 Creating Use Case Diagrams
- 2 Day 1: Learning the Basics of PL/SQL
- 3 C# Tip: Placing Your C# Application in the System Tray
- 4 Using ASP.NET To Send Email
- 5 Using JDBC with MySQL, Getting Started

Recommended Partner Resources

- Cloud Computing Showcase for Developers
- Mobile Development Center
- HTML 5 Development Center

Sitemap | Contact Us

Thanks for your registration, follow us on our social networks to keep up-to-date