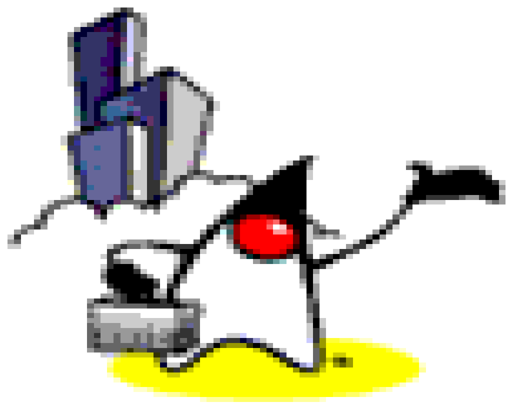
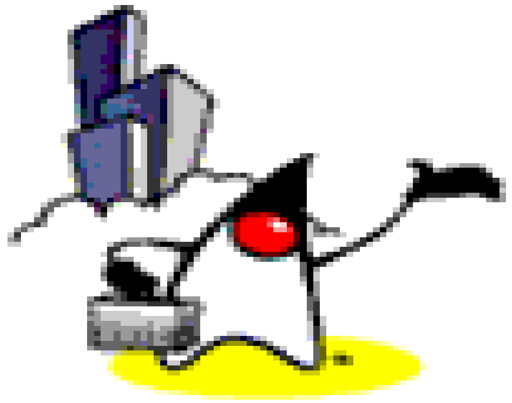


Hibernate Caching

Topics

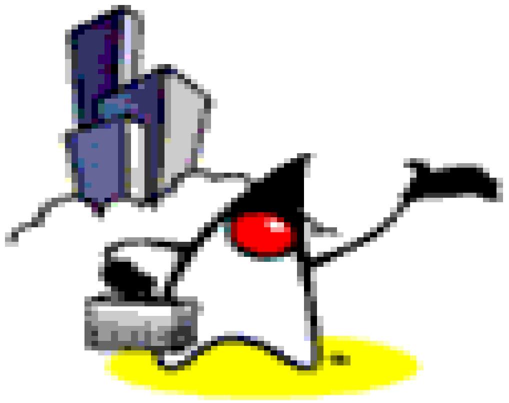
- What is Caching?
- Hibernate caching
- Second-Level Caching implementations
- Second-Level Caching strategies
- Hibernate performance monitoring



Caching

What is Caching?

- Used for optimizing database applications.
- A cache is designed to reduce traffic between your application and the database by conserving data already loaded from the database.
  - Database access is necessary only when retrieving data that is not currently available in the cache.
  - The application may need to empty (invalidate) the cache from time to time if the database is updated or modified in some way, because it has no way of knowing whether the cache is up to date



Hibernate Caching

Two Different Caches (for Caching Objects)

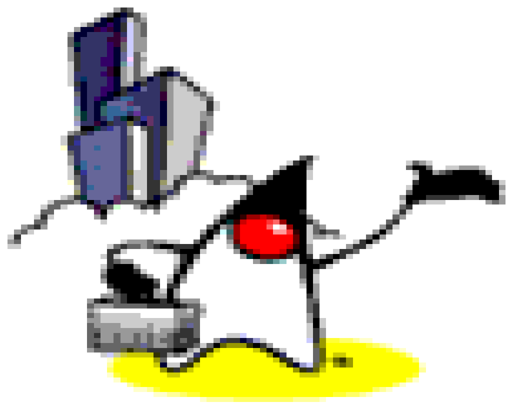
- First-level cache
  - associated with the *Session* object
- Second-level cache
  - associated with the *SessionFactory* object

First-level Cache

- Hibernate uses first-level cache on a per-transaction basis (within a single transaction boundary)
  - Used mainly to reduce the number of SQL queries it needs to generate within a given transaction.
  - For example, if an object is modified several times within the same transaction, Hibernate will generate only one SQL UPDATE statement at the end of the transaction, containing all the modifications

Second-level Cache

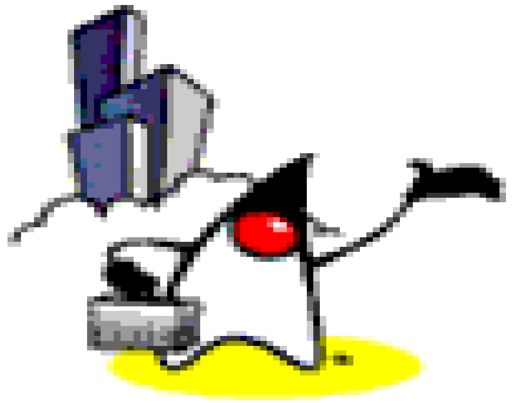
- Second-level cache keeps loaded objects at the Session Factory level across transaction boundaries
- The objects in the second-level cache are available to the whole application, not just to the user running the query
  - This way, each time a query returns an object that is already loaded in the cache, one or more database transactions potentially are avoided.



Query-level Cache

Query-level Cache

- Use it when you need to cache actual query results, rather than just persistent objects



Second-Level Cache Implementations

Cache Implementations

- Hibernate supports these open-source cache implementations out of the box
  - EHCache (org.hibernate.cache.EhCacheProvider)
  - OSCache (org.hibernate.cache.OSCacheProvider)
  - SwarmCache (org.hibernate.cache.SwarmCacheProvider)
  - JBoss TreeCache (org.hibernate.cache.TreeCacheProvider)
- Commercial cache implementation
  - Tangosol Coherence cache

EHCache

- Fast, lightweight, and easy-to-use in-process cache
- Supports read-only and read/write caching, and memory- and disk-based caching.
- However, it does not support clustering

OSCache

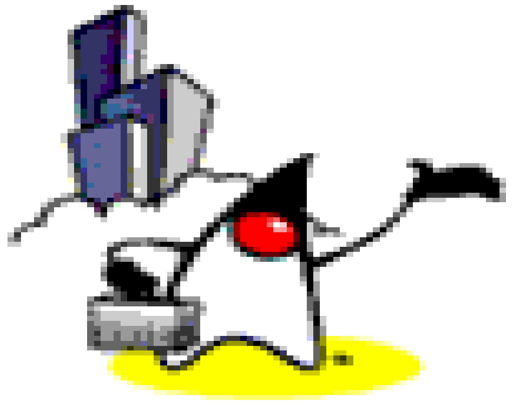
- Part of a larger package, which also provides caching functionalities for JSP pages or arbitrary objects.
- It is a powerful and flexible package, which, like EHCache, supports read-only and read/write caching, and memory- and disk-based caching.
- It also provides basic support for clustering via either JavaGroups or JMS

SwarmCache

- Is a simple cluster-based caching solution based on JavaGroups.
- Supports read-only or nonstrict read/write caching (the next section explains this term).
- This type of cache is appropriate for applications that typically have many more read operations than write operations.

JBoss TreeCache

- Is a powerful replicated (synchronous or asynchronous) and transactional cache.
- Use this solution if you really need a true transaction-capable caching architecture



Cache Strategies (for Second-Level)

Caching Strategies

- Read-only
- Read/write
- Nonstrict read/write
- Transactional

Caching Strategy: Read-only

- Useful for data that is read frequently but never updated.
- Simplest and best-performing cache strategy.

Caching Strategy: Read-only

```
<hibernate-mapping
    package="com.wakaleo.articles.caching.businessobjects">
   <class name="Country" table="COUNTRY" dynamic-update="true">
      <meta attribute="implement-equals">true</meta>
      <cache usage="read-only"/>

      <id name="id" type="long" unsaved-value="null" >
         <column name="cn_id" not-null="true"/>
         <generator class="increment"/>
      </id>

      <property column="cn_code" name="code" type="string"/>
      <property column="cn_name" name="name" type="string"/>

      <set name="airports" >
         <cache usage="read-only"/>
         <key column="cn_id"/>
         <one-to-many class="Airport"/>
      </set>
   </class>
</hibernate-mapping>
```

Caching Strategy: Read/Write

- Appropriate if your data needs to be updated.
- They carry more overhead than read-only caches.
- In non-JTA environments, each transaction should be completed when *Session.close()* or *Session.disconnect()* is called.

Caching Strategy: Read/Write

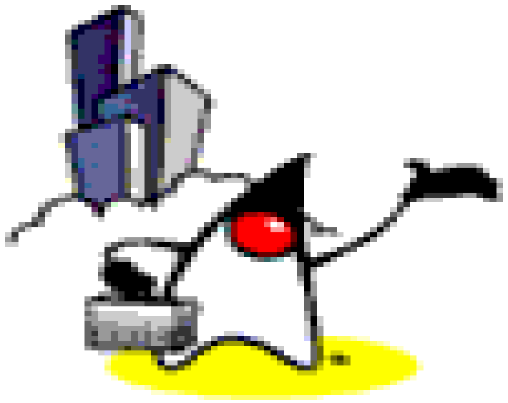
```
<hibernate-mapping package="com.wakaleo.articles.caching.businessobjects">
    <class name="Employee" table="EMPLOYEE" dynamic-update="true">
        <meta attribute="implement-equals">true</meta>
        <cache usage="read-write"/>

        <id name="id" type="long" unsaved-value="null" >
            <column name="emp_id" not-null="true"/>
            <generator class="increment"/>
        </id>
        <property column="emp_surname" name="surname" type="string"/>
        <property column="emp_firstname" name="firstname" type="string"/>
        <many-to-one name="country"
        column="cn_id"
        class="com.wakaleo.articles.caching.businessobjects.Country"
        not-null="true" />
        <!-- Lazy-loading is disactivated to demonstrate caching behavior -->
        <set name="languages" table="EMPLOYEE_SPEAKS_LANGUAGE" lazy="false">
            <cache usage="read-write"/>
            <key column="emp_id"/>
            <many-to-many column="lan_id" class="Language"/>
        </set>
    </class>
</hibernate-mapping>
```

Caching Strategy: Nonstrict Cache

- Does not guarantee that two transactions won't simultaneously modify the same data.
  - Therefore, it may be most appropriate for data that is read often but only occasionally modified.

Caching Strategy: Transactional

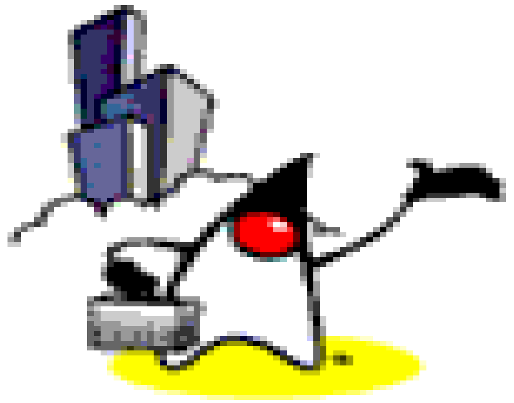
- A fully transactional cache that may be used only in a JTA environment



Cache Configuration

Cache Configuration in Hibernate Configuration File

```
<hibernate-configuration>
   <session-factory>
...
        <property name="hibernate.cache.use_query_cache">
            true
        </property>
        <property name="hibernate.cache.use_second_level_cache">
            true
        </property>
        <property name="hibernate.cache.provider_class">
            org.hibernate.cache.EhCacheProvider
        </property>
...
   </session-factory>
</hibernate-configuration>
```



Monitoring Performance

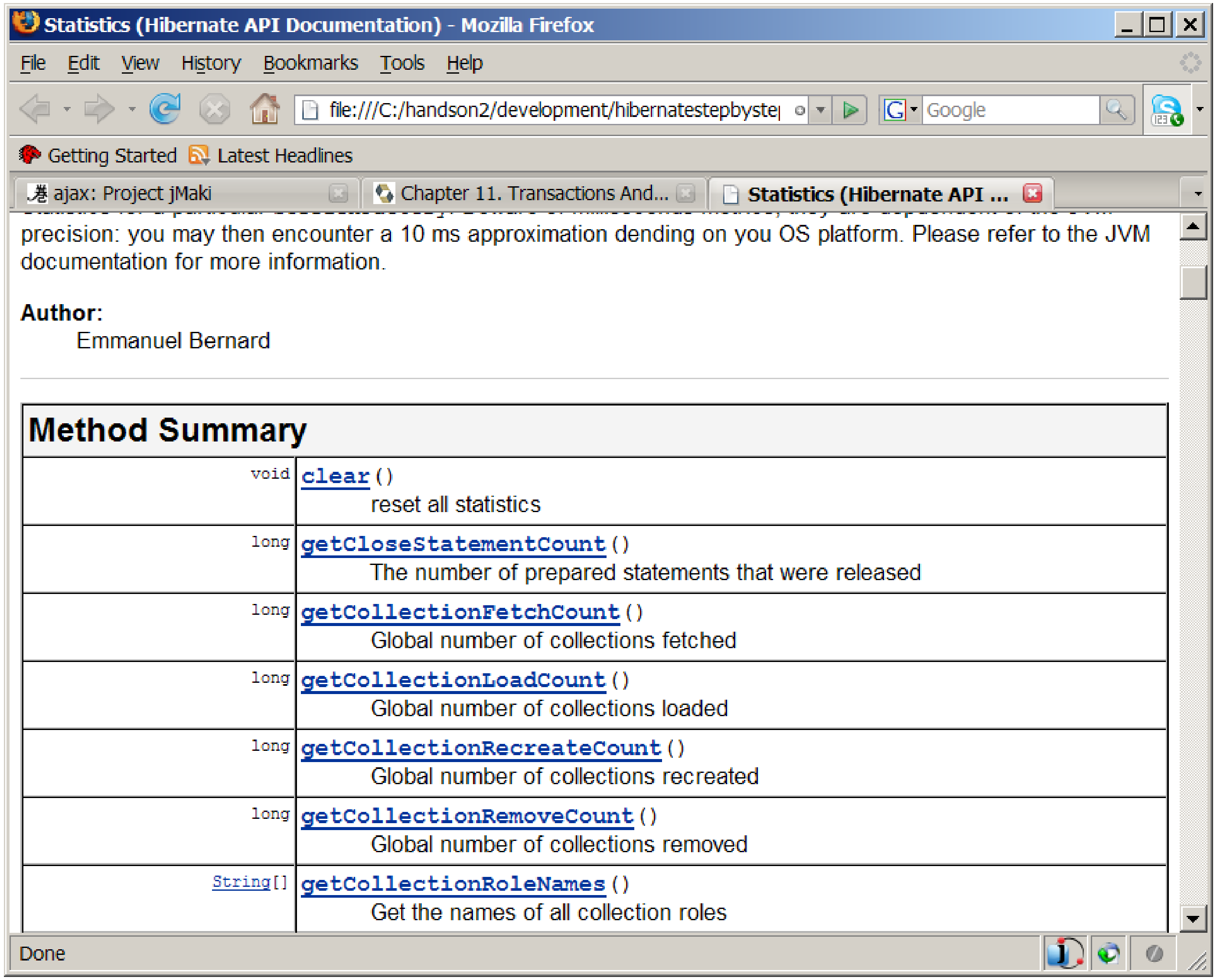
Statistics

- Hibernate provides a full range of figures about its internal operations.
- Statistics in Hibernate are available per *SessionFactory*.
  - Option1: Call *sessionFactory.getStatistics()* and read or display the Statistics yourself.
  - Option2: Through JMX

Statistics Interface

- Hibernate provides a number of metrics, from very basic to the specialized information only relevant in certain scenarios.
- All available counters are described in the Statistics interface API, in three categories:
  - Metrics related to the general Session usage, such as number of open sessions, retrieved JDBC connections, etc.
  - Metrics related to he entities, collections, queries, and caches as a whole (aka global metrics),
  - Detailed metrics related to a particular entity, collection, query or cache region.

Statistics Interface



Statistics (Hibernate API Documentation) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/handson2/development/hibernatestepbyste|
Google
Getting Started
Latest Headlines
ajax: Project jMaki
Chapter 11. Transactions And...
Statistics (Hibernate API ...
Done

precision: you may then encounter a 10 ms approximation dending on you OS platform. Please refer to the JVM documentation for more information.

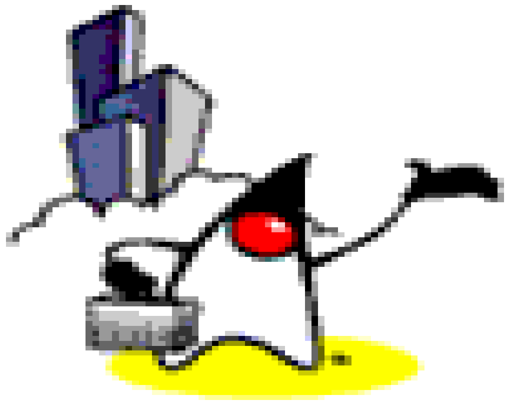
Author:
Emmanuel Bernard

Method Summary

void	clear() reset all statistics
long	getCloseStatementCount() The number of prepared statements that were released
long	getCollectionFetchCount() Global number of collections fetched
long	getCollectionLoadCount() Global number of collections loaded
long	getCollectionRecreateCount() Global number of collections recreated
long	getCollectionRemoveCount() Global number of collections removed
String[]	getCollectionRoleNames() Get the names of all collection roles

Example Code

```
Statistics stats = HibernateUtil.sessionFactory.getStatistics();

double queryCacheHitCount  = stats.getQueryCacheHitCount();
double queryCacheMissCount = stats.getQueryCacheMissCount();
double queryCacheHitRatio =
  queryCacheHitCount / (queryCacheHitCount +
    queryCacheMissCount);

log.info("Query Hit ratio:" + queryCacheHitRatio);

EntityStatistics entityStats =
  stats.getEntityStatistics( Cat.class.getName() );
long changes =
        entityStats.getInsertCount()
        + entityStats.getUpdateCount()
        + entityStats.getDeleteCount();
log.info(Cat.class.getName() + " changed " + chang
```



Thank You!