Batch Lazy Loader

Ryan Senior
Source Allies Inc
4001 Ingersoll Ave
Des Moines, IA
senior.ryan@gmail.com

ABSTRACT

Retrieve many related, Lazy Loaded objects simultaneously, overcoming the performance degradation that can be associated with the Lazy Load pattern (called *ripple loading* by Fowler [1]).

1. MOTIVATION

Often times in Domain Models, not all of the relationships associated to a particular object are needed all the time. The Lazy Load pattern is often used to help improve the performance of relationships in the model that are not always needed. Lazy Loading improves the performance of the application by delaying the retrieval of certain pieces of data until it is needed. This will allow the program to save resources if the data is not needed. There are, however, certain situations when the Lazy Load pattern can severely degrade performance. The application may not always need instances of that object, so Lazy Loading may still make sense. But for the few times that the Lazy Loaded objects are needed, the loading could cause major performance problems. Typically these performance problems are due to an excessive number of requests required to individually retrieve each Lazy Loaded piece of data. In the context of a database, this could cause an excessive number of queries to be executed (since a separate query would be required for each Lazy Loaded association that was accessed).

To illustrate this example, let's assume that there is an application that maintains employee information for a company and generates paychecks for the employees of the company. There's a lot of information associated to an employee, their address, their time-card for a week, pay, etc. Let's assume that during most of the day to day operations of this program, the address information is not used. This makes the address association a good candidate for the Lazy Load pattern. The class diagram would look something like Figure 1.

In this example, we'll assume we're using a simple Domain Model. This Domain Model will be backed by a relational database and use a single table per class design. The core of this database structure is the Employee table. The Salary table contains a foreign key reference to the Employee table and the Timecard table also contains a foreign key relating it to the Employee table. In the Domain Model above, Salary-Info and Timecards are accessed via an instance of the Employee class and are non-lazy. That means that when an Employee object is retrieved (i.e. a row in the Employee table is retrieved from the database) then the associated SalaryInfo instance and Timecard instance are also retrieved. The Address relationship is Lazy Loaded, so the Address instance won't be retrieved until the getAddress() method on Employee is called. Suppose it's the 15th of the month and it's paycheck time. The application must know where to send the paycheck, so it will need address information for each employee. The database is queried and a collection of five hundred employees are retrieved. The application then processes each employee's time-card and pay. Once the amount of pay is calculated, the getAddress() method is called to get the address of the employee. Each time this method is called, a query is sent to the database and the data is retrieved for one address. So in our example, one query retrieves the five hundred employees, but then, five hundred separate queries are executed to retrieve address information. Executing five hundred queries will most likely be very costly in the execution of the program when compared to the single query that retrieves the five hundred employees. Also, executing five hundred queries is probably not what the developer intended, but is a down-side of using the Lazy Load pattern. One option to correct this problem is to disable Lazy Loading of the address object. This would probably be the easiest way to fix the problem, however there could be a degradation of performance in the rest of the application. Once Lazy Loading is disabled, every Employee read from the database will also cause an Address to be read (along with the SalaryInfo and Timecard instances). This would be a waste of system resources in our situation because Address is not needed most of the time. If disabling the Lazy Loading of the information is not a good option, we can use the Batch Lazy Loader pattern. The Batch Lazy Loader pattern allows the existing portion of the application that uses Lazy Loading to continue using it. In the cases that many Lazy Loaded relationships are requested, batching the Lazy Load calls into one, can significantly improve performance. This allows the performance improvement of the Lazy Load pattern while minimizing the drawbacks of the pattern due to ripple loading.
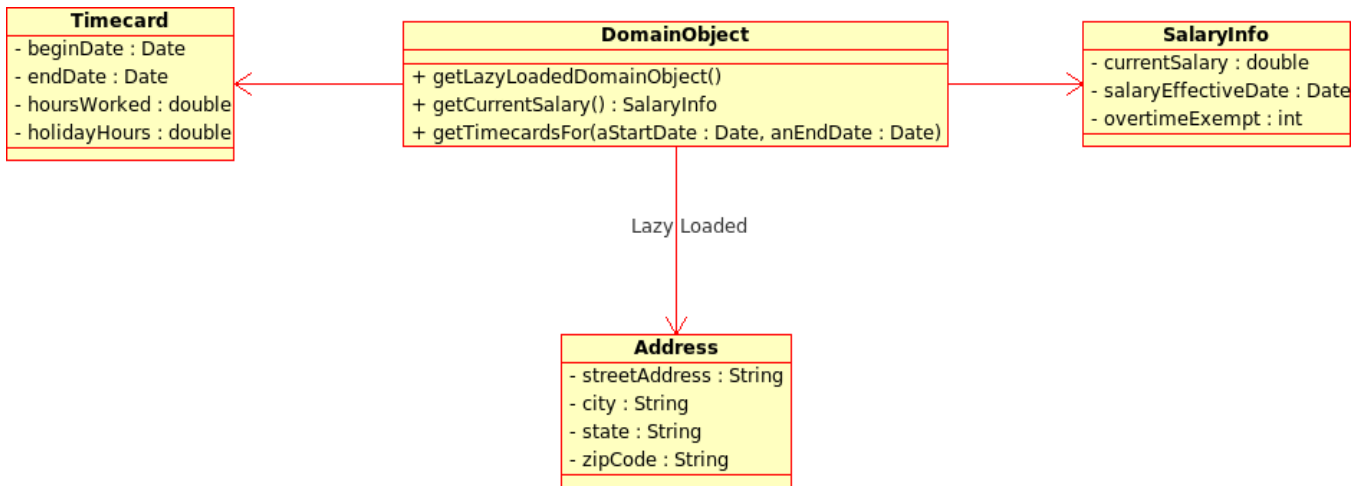
Figure 1: Class diagram of the Employee class and its associations.

2. SOLUTION

First create some flag to indicate that a Lazy Loaded Domain object is batch loadable (like the Address class in the above example). This could be a property setting (in a configuration file), an attribute in the source of the class, an annotation or maybe an additional method call setting a flag before retrieving the data. When loading an object that references a Batch Lazy Loadable object, it is included in a list. Though it is in this list, the batch loaded data is not actually retreived from the source. Then, when any one of the addresses are requested, the rest of the addresses in the batch read list are also retrieved. There are two approaches to batch reading this information. Oracle's Toplink takes the approach of allowing developers to read in all of the pending Lazy Loaded Domain objects when any one of those Lazy Loaded objects are requested. In the Employee example above, all five hundred employee addresses would be retrieved whenever the first Employee's getAddress() method was invoked. The first getAddress() method invoked doesn't have to be from the first Employee chronologically, this is just the first time the getAddress() method is called on any of the Employees loaded from the database. In the above example, this could be the 1st record or the 10th or 30th etc. Another approach (the one used in Hibernate) is to specify the number of records to batch retrieve. Assuming the batch size is set to 100, when the first Address of the first Employee is read from the database, a query will execute retrieving one hundred addresses. When the 101st employee and 101st address is accessed, a second query will be initiated, bringing in the next one hundred records from the database.

This can be implemented different ways depending on the approach chosen above and how the data is being Lazy Loaded in the application. One common thread between the two options is the use of a type of Identity Map. An Identity Map can be thought of as a cache for rows in a database table. The Identity Map contains a series of data structures, aimed at caching data retrieved from the database. Each row is considered unique and stored in the structure. When a row is requested that is already in the Identity Map, it is just retrieved from the Identity Map instead of incurring the overhead of retrieving it from the database again. When

an element is not in the Identity Map it is retrieved from the database and then placed in the map. The usage of the Identity Map pattern is slightly different in the context of the Batch Lazy Loader. The concern isn't the caching of the records from the database, but the recording of objects that need to be Lazy Loaded. So this Identity Map is populated with information about an instance of a class in the Domain Model and it's Lazy Loaded association. These "to-be" Lazy Loaded references are stored in that map until finally one of the Lazy Loaded instances is needed. Then that map is queried for the instances of that Lazy Loaded object that need to be retrieved and then the query is sent for the data.

3. APPLICABILITY

Use the batch Lazy Loading pattern when:

- The Lazy Loaded object is not able to be loaded via a non-lazy means (such as when performance demands it)

- A large number of the same type of Lazy Loaded objects are needed and performance demands that they be loaded in a more non-lazy fashion

4. STRUCTURE

The structure of this implementation will vary greatly depending on how the Domain Model, Lazy Loading and existing Identity Maps are implemented. See Figure 2 for a sequence diagram of the flow of events when using Batch Lazy Loading.

Participants:

- Domain Object

  - contains information related to an entity in the domain model

  - has a relationship to a Lazy Loaded object from the domain model
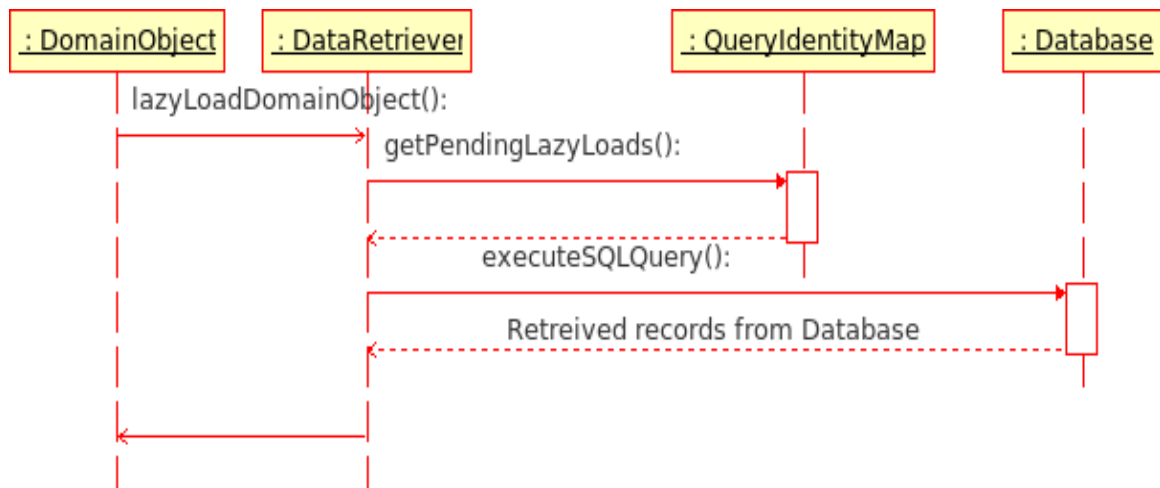
- DataRetriever

Figure 2: Class diagram of the Employee class and its associations.

- determines whether an object needs Lazy Loading or is already in memory
- calls to the QueryIdentityMap to determine if other elements of the same type need to be loaded and includes them in the query for the currently loading object
- retrieves the Lazy Loaded data

- QueryIdentityMap
  - maintains a list of objects to Lazy Load may also
  - contain a predefined number of objects to load at a given time

5. COLLABORATIONS

- Application (or user) requests a Lazy Loaded Domain Object

- DataRetriever consults the QueryIdentityMap to obtain objects to Lazy Load

- DataRetriever will need to collaborate with many classes to retrieve data from the database

6. CONSEQUENCES

Using the Batch Lazy Loader pattern leads to increased performance versus Lazy Loading of a large number of Domain Objects. Whenever attempting to improve the performance in an application, it is crucial to get a benchmark. How long did a particular process or action take when the Domain Objects were Lazy Loaded? How long after Batch Lazy Loading them? The biggest performance gains typically occur when there is significant overhead associated with requesting and/or retrieving the data. This overhead could come from making the connection to the remote source, compiling/interpreting the requested query and/or the overhead of actually executing the query. In this case, batching the calls together could incur that overhead just once (or a few times), instead of many times.

There are also situations in which the Batch Lazy Loader pattern can decrease performance. One such case is batch retrieving a very large amount of data. Batch loading will cause all of the Lazy Loaded Domain Objects to be in memory same time. This will require much more space than when they were individually Lazy Loaded, which could put unnecessary strain on the application. Another case of degraded performance would be if not all of the Batch Lazy Loaded elements were used. If, for example, five hundred Lazy Loaded elements were retrieved and only ten were used. Since most of the data would be unused, it could degrade performance and would defeat the purpose of the Lazy Load pattern.

7. IMPLEMENTATION

1. In Object to Relational Mapping tools, often there is a flag or other indicator that sets a particular relationship to be always batch loaded or not

2. Hibernate specifies the number of records to batch Lazy Load in an XML configuration file

3. Toplink also allows notification that a Lazy Loaded object should be batch loaded at runtime, through a query parameter

4. Often the DataRetriever above is not a single class but a group of classes that handle retrieving data

8. KNOWN USES

This pattern is used in many Object to Relational Mapping tools like Hibernate and Toplink.

9. RELATED PATTERNS

Domain Model - Batch Lazy Loader will often be used with a Domain Model

Identity Map - Used to keep track of which elements to batch Lazy Load

Lazy Load - Batch Lazy Loader is needed as a side affect of the Lazy Load pattern

10. REFERENCES
[1] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.