Design Specification Document

Version 1.0.0

Server-Side Cache

N-Way Set Associative Cache

February 17, 2018

Introduction

This is a design specification document for Server-Side Cache. This library enables developers to make their application faster and to reduce the bottleneck to database system.

This software is a java library, which can be integrated in any Java application. This java library is a caching system, which enables developers to cache any kind of data in memory for their application. The architecture used for this caching system is N-way Set Associative Cache. Caching reduces the number of hits to the database system, thus significantly improving the performance of application and reducing bottleneck to database.

Architecture

The software implements N-way Set Associative Cache architecture. To know more about this architecture, click here.

Design

Data Structures Used:

DataBlock – This represents the data block in cache.

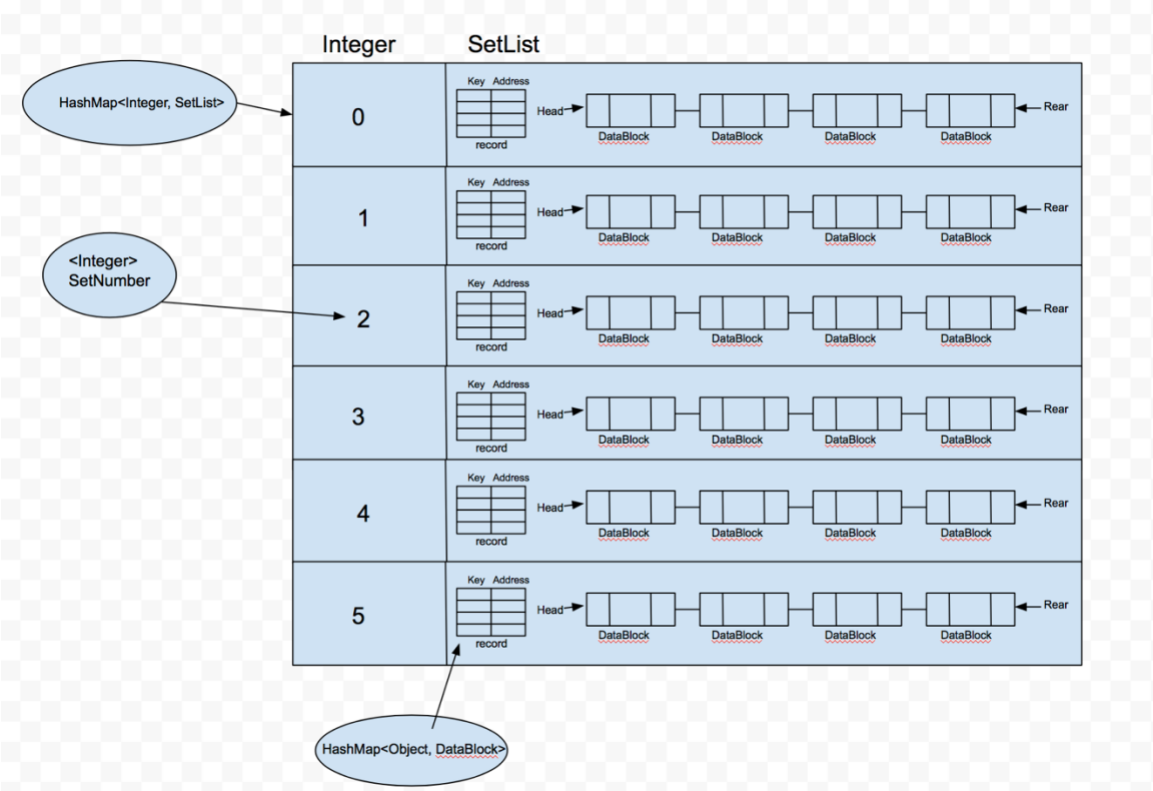
Doubly Linked List – Represents the list of DataBlocks in the set. This doubly linked list data structure also contains a hashmap, which maintains records of datablock present in this set.

HashMap – Represents N-way set associative cache, with multiple sets in it. (Number of sets is equal to size of hashmap)

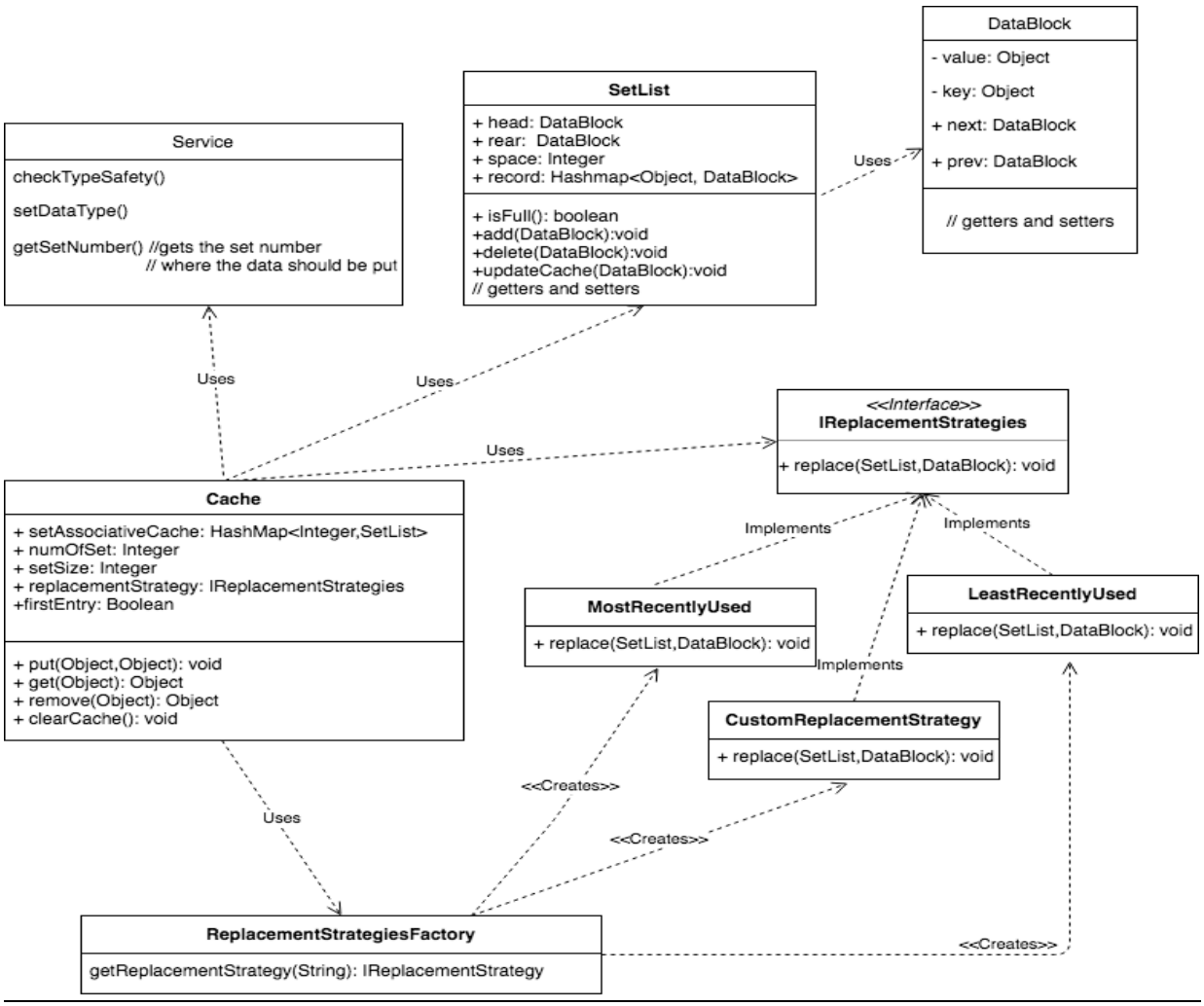
Data Structure Usage Explained:

- Cache is represented using HashMap<Integer,SetList>, where Integer represents the set number, and SetList represents the List of DataBlocks.
- Each SetList represents the List of DataBlocks in that set. Maximum size of SetList can be N, since it is N-way set associative cache.
- Each SetList also contains a HashMap<Object,DataBlock>. This hashmap maintains key and corresponding address of DataBlock.

Data Structure Visualization:



Class Diagram:



Note: For more clear view, please check Class Diagram.pdf

Implementation Strategy

Cache:

Initialization:

Cache can be initialized by specifying number of sets, N, Datatype for keys and values. If not specified, then 2 – way set associative cache with 10 sets will be initialized.

Put:

When put operation is called on cache, it puts DataBlock in the cache. Here, the set in which the DataBlock will go is decided using inbuilt hash function. Once the set, where DataBlock is to be inserted is decided, the DataBlock is inserted to that particular set. If the key for this DataBlock is already present in the cache, then the old datablock is updated with new value.

If the cache is full, and put() method is invoked on cache, then the replacement strategy is called to replace one of the DataBlock with new DataBlock.

Get:
When get operation is called on cache, the function expects the key for the DataBlock to be retrieved. If the key is not present in the cache, "null" is returned by the method. If key is present in the cache, corresponding value will be retrieved. Whenever, a get is called on particular key, the location of corresponding block in the SetList is updated.

Remove:
When put operation is called on cache, the function expects the key for the DataBlock to be removed. If the key is not present in the cache, "null" is returned by the method, and Cache remains unaffected.

Note: The SetList is always sorted according to the last access. i.e. The element most recently accessed will always will be at the head of SetList. The element least recently used will always be at the rear of SetList.

Replacement Strategies:
This library provides 2 replacement startegies:

1. Least Recently Used
   The oldest DataBlock is removed and new DataBlock is added.
2. Most Recently Used
   The most recently added/updated DataBlock is removed.

Service:
Service is a singleton class, which provides different service to implement the Cache.class. It provides 2 function:

1. getSetNumber()
   This function generates the hash for the input, and scales it to a value between 0 and number of sets.
2. checkTypeSafety()
   This function checks whether the data types given by user are safe or not. i.e. for a given instance of cache, the datatypes of keys and values should be same.

Logging:
Library used for logging is Log4J. Logger logs "info" and "error" statements in the log file. The current log file will always be named as "log.out". All log files are saved on daily basis. Past log files will be stored with date appended at the end of file name like "log.out.2018-02-15".

Logger properties can be configured using log4j.properties file.

Testing:
Testing is done using Junit testing framework. Nomenclature used for test class is by using suffix “Test” after the class name. All the test classes are present in “test” package and can be executed by using “testCache” run configuration. They can also be executed individually.

API Access

To integrate this cache system with your application, you need below file:
N-waySetAssociativeCache.jar

Once this file is downloaded, the .jar file can be added to the application by adding it to classpath.

You can download JAR file here.

How to add .jar file to your project?

Eclipse:

1. Start Eclipse and locate the project folder to which this Cache library should be added.
2. Right-click this class folder, and select "Properties"
3. Select "Java Build Path" on the left, and then the "Libraries" tab. Now, click the "Add External JARS..." button
4. Locate and select the "N- waySetAssociativeCache.jar" file you just downloaded, and then click "Open"
5. Finally, click "OK" to close the dialog box. You will know that everything went ok if, when you open your project folder, you see an item called "Referenced Libraries", and upon expanding this item, you see the package " N-waySetAssociativeCache.jar" listed.

Command Line:

You can execute the command below: (The command will only work if you are using Java6 or above)

```
Java –classpath /classes;/out/N_waySetAssociativeCache_1_0_0_jar/ N-waySetAssociativeCache.jar
```

Functionalities

Initialize: It initializes cache
parameters:

numOfSet – number of sets
setSize – 'N' in N way set associative cache
Keytype – Datatype of key
Valuetype – Datatype of value
Strategy – Replacement Strategy (lru,mru,custom)

Example Initialization:

```
Cache c = new Cache(1,3,Integer.class,String.class,"mru");
```

```
Cache c = new Cache(1,3,Integer.class,String.class,"lru");
```

```
Cache c = new Cache(1,3,Integer.class,String.class,"custom");
```

Note: We need to pass the class of datatypes to constructor in-order to Set the datatypes of Key and Value.

Put: To put the key and value in cache
Parameters: Key and Value
Input Datatype: As set while initializing cache.
Put operation should always be in try-catch

```
//put should always be in try-catch
try {
    c.put(5, "Test");
}catch (TypeMismatchException e){
    e.printStackTrace();
}
```

Get: To Get value in from cache, corresponding to key provided
Parameters: Key
Input Datatype: As set while initializing cache.
Get operation should always be in try-catch

```
try {
    c.get(5);
}catch (TypeMismatchException e){
```

```
    e.printStackTrace();
}
```

Remove: To remove DataBlock from cache, corresponding to key provided
Parameters: Key
Input Datatype: As set while initializing cache.
Remove operation should always be in try-catch

```
//remove should always be in try-catch
try {
    c.remove(5);
}catch (TypeMismatchException e){
    e.printStackTrace();
}
```

ClearCache: Clears the entire cache
Parameters: None

```
c.clearCache();
```

Providing Custom Replacement Strategy:

Client can provide custom strategy for replacement when cache is full. This can be done by extending "CustomReplacementStrategy.Class". The replacement strategy logic will go in method " replace(SetList, DataBlock)".

This method takes 2 parameters as input.
1. SetList – Doubly linked list, with hashmap representing a set in cache.
2. DataBlock – DataBlock to be added

Important Note:

1. The SetList is always sorted according to the last access. i.e. The element most recently accessed will always will be at the head of SetList. The element least recently used will always be at the rear of SetList.
2. To enable logger, call super() as first statement from remove(SetList, DataBlock) method.

Example:

```
public class MyReplacementStrategy extends CustomReplacementStrategy {
```

```
    @Override
    public void replace(SetList list, DataBlock block) {
        super.replace(list, block);

        //your code goes here
    }
}
```

Example Usage:

Below is the consolidated code, demonstrating usage of all the function.

```
//this is a sample Main() to learn more on How to access the API
public static void main(String[] args) {

    //Initialising the cache
    //Key data-type => Integer.class
    //Value data-type => User.class
    Cache cache = new Cache(1,2,Integer.class,User.class,"lru");


    //Creating Users
    User user1 = new User(1,"User1","Lname1");
    User user2 = new User(2, "User2","Lname2");
    User user3 = new User(3, "User3","Lname3");


    try {
        //putting the users in cache
        cache.put(1, user1);
```

```
        cache.put(2, user2);
        //Get u2
        cache.get(2);
        //Now Set1 in cache is full
        //On next put u2 will be replaced by u1
        cache.put(3, user3);
        //Now our cache is u3->u1
        cache.get(1);
        //Now our cache is u1->u2

        //removing the u1 from cache
        cache.remove(1);

    }catch (TypeMismatchException e){
        System.out.println(e.getMessage());
    }


}
```

Dependencies

1. Log4J
2. Junit