## hqrank介绍

支持并发的实时排行、排行榜工具-java

1、目前hqrank的实现包括如下几点：

（1）全缓存：所有的排行数据存储在内存中。所以，hqrank是作为一个排行工具，而不是数据存储工具而存在

（2）基于id:value：对数据的存取可以通过id-value进行，可通过id获取对应排名，也可以通过排名获取对应id（value也会在此给出）。分数不同的按照从大到小的顺序排列，分数相同的按照插入或修改时间顺序排列

（3） 支持多字段排行：比如先按等级排行，在此基础上再按经验排行，进而得出的排行列表。理论上支持任意多的字段数

（4） 支持并发：支持多个线程同时对排行进行操作

（5） 实时排行：排行操作是实时进行的，但需要说明一点：对于访问时间相差很少的两个访问，尤其是不同线程，可能不会严格满足先入先排，但这种相差时间是在毫秒级的（这和线程时间片以及系统提供加锁顺序相关）

2、目前实现的取数据功能

（1）根据id获取排行数据及分数

（2）根据排行获取对应id

（3）根据id获取id前后若干个玩家的排行数据

（4）分页查询，根据每页大小和页码获取排行数据

（5）根据字段更新数据

## 如何使用

hqrank/code/Rank/是一个java项目，开发使用jdk版本为：1.7

（1）将该项目导入eclipse中

（2）在包org/hq/rank/service/中有IRankService.java，里面有详细的注释

（3）基本访问方法如下：（[[点击查看所有接口方法]](https://github.com/xuerong/hqrank/blob/master/code/Rank/src/org/hq/rank/service/IRankService.java)）

```Java

public static void main(String[] args) {

IRankService rankService = new RankService();

rankService.createRank("rankName");

rankService.put("rankName", 10/\*id\*/, 100/\*value\*/); // put date to rank

RankData rankData = rankService.getRankDataById("rankName", 10); // get date by id

int rankNum = rankData.getRankNum(); // get rank num

RankData rankData2 = rankService.getRankDataByRankNum("rankName", rankNum); // get date by rankNum

List<RankData> rankDataList1 = rankService.getRankDatasAroundId("rankName", testId, 3, 6); // get date by id,and ranks around this id

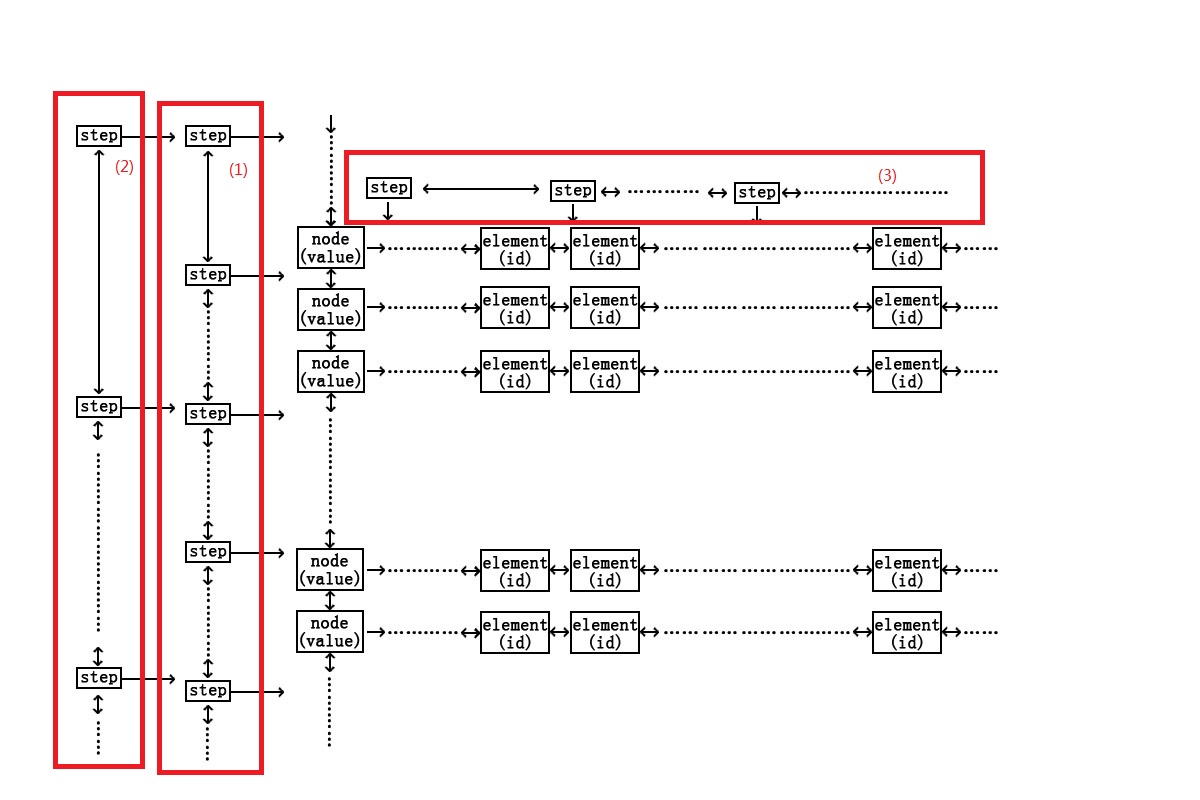
List<RankData> rankDataList2 = rankService.getRankDatasByPage("rankName", 7/\*page\*/, 9/\*pageSize\*/); // get date by page

}

```

## hqrank原理

1、排行数据结构



（1） node分为elementNode和rankElementNode，上诉图中的node为elementNode

（2）elementNode链表按照从大到小记录所有value,每个elementNode节点记录一个分数，保存一个element节点链表

（3）每个element链表按照时间从小到大记录该value的id，每个element节点记录一个id

（4）step链表(1)是对node链表的索引，提高对node的定位速度

（5）step链表(2)是对step链表(1)的索引，提高对step(1)的定位速度，从而提高对node的定位速度

（6）step链表(3)是对element链表的索引，提高对element的定位速度

（7）step和node中保存有其索引范围内的element数量，即排行所需数据

（8）对于多字段查询，除了最后一个字段层级使用elementNode用于存储element，其余字段层级使用rankElementNode，每个rankElementNode保存一个新的rank

2、并发和加锁

（1）element是最基本的存储单位，每一个element自身拥有一个AtomicInteger作为锁，用原子加减操作作为加解锁

（2）对于添加和修改数据，都会创建新的element，并对其先加锁，后加入到数据容器中

（3）对于element在数据容器中的增加和删除都会对其前后element进行加锁

（4）对于node和rankElement则使用锁池LockerPool

（5）对于step，当达到最低索引数量时，才会被创建，每一个step对象拥有自身的锁对象

（6）整个结构形成了nodeStep1-nodeStep2-node-elementStep-element的结构，对于每一层级的操作仅对该层级的对象加锁，同时对多个层级操作需要对多个层级加锁（如因为node数量增加而创建nodeStep），每个层级的锁尽量在本层级释放

（7）查询不加锁，为保持效率和正确性，采取若查询失败，可配置多次查询直到获取所得值或达到最大查询次数

（8）重新执行方案：当一个操作，包括对数据的增删改和对node的删除，加锁失败时，不等待锁，而是放入reoper池，重新执行。 重新执行时间和最多执行次数由工具参数配置决定

3、关于性能的讨论

（1）在node节点作为数据存储元之上是一个两层索引的跳表，而element链表上是一个一层索引的跳表

（2）在后面的设计中可以对跳表的层数以及每层的索引数进行设计，但要注意以下几点：第一，索引层级越多，每次修改数据对索引的操作就会越多，第二，索引越多，对存储空间的要求越高，第三，为支持并发，索引越多，加锁要求增加

（3）如果采取不支持并发的策略（或通过访问排队来支持并发），一来是不能使用多核cpu的优势，二来当使用同步持久化的时候效率会骤降

（4）实测(MacBook Pro,Core(TM) i7-3520M CPU @ 2.90GHz 双核（四逻辑核心）,DDR3 8G 1600HZ，-Xmx4096m):1000万数据排行，200线程共每分钟访问50万（均匀访问）排行请求，cpu利用率维持在100%（共400%），访问复杂度基本不变，注意：当内存即将耗尽，CPU会更加多的走高，直到资源耗尽（gc频率增加）

（5）目前每一个排行数据需要消耗高达280B的内存，使得系统对内存要求较高

4、其它

1、工具所配置参数对象RankConfigure

2、排行异常(目前还没有具体实现)RankException

3、数据校验和统计RankStatistics：包括对数据结构中各个对象的统计，锁统计，reoper统计，访问统计，step统计，以及数据正确性校验

4、对象池RankPool，排行所用到的数据对象，包括node、element、rankElement、nodeStep、elementStep，通过对象池来管理，以避免过多的创建对象

## 更多的说明

1、redis不是hqrank项目中的部分，是用来测试的，因为redis中有实时排序功能，在这里也提供一个redis的下载[Redis-x64-2.8.2400](http://pan.baidu.com/s/1o87v5s2)

2、test包中有基本的例子。BaseTest1：基本的用法；BaseTest2：压力测试

3、测试过程中可以用jprofiler和jdk工具jconsole.exe来监控内存使用情况

4、当数据量较大时，需要设置较大的JVM内存大小

5、工具还在开发过程中，还没有经过大量测试，如果有对此有兴趣的Coder，欢迎一起讨论和开发，更多的开发计划在\_packet\_.java文件中简述，目前有：

## 后面想要实现的功能

1、配置文件：针对所有的rank，针对单个的rank（配置rank）

2、添加持久化：(1)同步/异步存储到文件/数据库，(2)启动加载，停止存储

3、失败的操作持久化

4、统一所有的异常，在service层捕获并处理

5、异常系统

6、添加英文注释？

7、添加AOP特性，对统计，异常等提供低耦合支持

8、添加网络接口访问

9、修改为maven项目？