1. **Concurrenthashmap**
2. 首先是其和hashtable,二者都是线程安全的，都可以用于多线程的环境，不同的是hashtable锁的是整个map，不适合高并发的系统，它仅有单个锁，对整个集合加锁。而ConcurrentHashMap仅仅锁定map的某个部分
3. 和JDK1.7相比，JDK1.8的ConcurrentHashMap由原来的数组+链表变为数组+链表+红黑树，这点和hashmap一样，但是加锁方法不一样了，1.7时ConcurrentHashMap的锁是用segment锁住其对应的段数据（其实就是该段对应的hashmap，感觉ConcurrentHashMap就是hashmap外层再加上个数组），JDK1.8的ConcurrentHashMap采用数组+链表+红黑树的实现方式来设计，内部大量采用CAS操作为对每个数组元素加锁
4. 1.7和1.8的扩容，1.7的扩容是对所有元素都重新进行hash算法再次存放，而1.8 中简化了这个操作，扩容后的数据要么在原来的位置再移动2次幂的位置，关于2次幂这个问题，数组每次扩充就会变为原来的2倍，可能是比较好算吧，每次只要左移一位就行。
5. 大体上就先总结这些，具体的put方法和链表以及红黑树之间的转换没有详细写出来。主要就是了解了1.7和1.8的hashmap和ConcurrentHashMap的不同以及前后变化的点。（了解了hashmap之后，终于知道为什么面试老问这个了，这简直是个超级大坑，任何一点逮着你就能一直问下去，真让人头大！）
6. r**edis开机自启动配置**

redis-server.exe --service-install redis.windows.conf --loglevel verbose

添加Redis服务redis-server --service-start开启服务

1. **mongo开机自启动配置**

mongod.exe --logpath D:\mongodb\logs\mongodb.log --logappend --dbpath D:\mongodb\data --directoryperdb --serviceName MongoDB --install

1. **通过redis实现5分钟内登录失败次数限制（3次失败，不允许登陆）（有多种实现方法）**

@RequestMapping("/login")

public String login(String id,String name) {

Set<String> keys=template.keys("\*");

if("1".equals(id)&&"admin".equals(name)) {

template.delete(keys);

return "success!";

}else {

//String key=name+"\_";

if(keys!=null&&keys.size()>=limit) {

count=0;

return "5分钟内只能操作3次，请五分钟后再试！";

}else {

count++;

System.out.println("失败"+count+"次");

String str=name+"\_"+count;

template.opsForValue().append(str,id);

template.expire(str, 300, TimeUnit.SECONDS);

return "失败"+count+"次";

}

}

}

1. **实现mongodb的map-reduce（不同页数的书籍的汇总）**

public class TestMongoMR {

public static void main(String[] args) {

Mongo mongo;

try {

mongo = new Mongo("localhost", 27017);

DB db = mongo.getDB("test");

DBCollection books = db.getCollection("books");

BasicDBObject book = new BasicDBObject();

book.put("name", "Understanding JAVA");

book.put("pages", 100);

books.insert(book);

book = new BasicDBObject();

book.put("name", "Understanding JSON");

book.put("pages", 200);

books.insert(book);

book = new BasicDBObject();

book.put("name", "Understanding XML");

book.put("pages", 300);

books.insert(book);

book = new BasicDBObject();

book.put("name", "Understanding Web Services");

book.put("pages", 400);

books.insert(book);

book = new BasicDBObject();

book.put("name", "Understanding Axis2");

book.put("pages", 150);

books.insert(book);

String map = "function(){ emit(this.name, this.pages); }";

String reduce = "function( key, values ){ return Array.sum(values); }";

MapReduceCommand cmd = new MapReduceCommand(books, map, reduce,

null, MapReduceCommand.OutputType.INLINE, null);

MapReduceOutput out = books.mapReduce(cmd);

for (DBObject o : out.results()) {

System.out.println(o.toString());

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

1. **实现mongo的圆形范围内的地理位置搜索**

**Dao层实现接口**

@Override

public List<Document> withinCircle(String collection, String locationField, Point center, long redius,

DBObject fields, DBObject query, int limit) {

// TODO Auto-generated method stub

if(query==null)

query = new BasicDBObject();

System.out.println("withinCircle:{}"+query.toString());

Bson bson = Filters.and(Arrays.asList(

Filters.geoWithinCenter(locationField, center.getX(), center.getY(), redius)));

FindIterable findIterable = mongoTemplate.getCollection("test").find(bson);

List<Document> result = new ArrayList<Document>();

MongoCursor<Document> mongoCursor = mongoTemplate.getCollection(collection).find(bson).limit(limit).iterator();

MongoCursor<Document> mongoCursor1 = findIterable.iterator();

while(mongoCursor.hasNext()){

result.add(mongoCursor.next());

}

return result;

}

}

**Controller层实现**

@RequestMapping("/withinCircle")

public List<Document> withinCircle(){

DBObject query = new BasicDBObject();

DBObject field = new BasicDBObject();

Point point=new Point(50, 50);

Long redius=50L;

int limit = 3;

List<Document> listfinal = userService.withinCircle("location", "loc", point, redius, field, query, limit);

for(Document obj : listfinal)

System.out.println(obj);

return listfinal;

}