t\_article(id int, title varchar, hits int)

1. 多个线程操作同一篇文章t\_article，怎样完成点击数累加？

乐观锁(ALTER TABLE ` t\_article ` ADD COLUMN `version` int)

**while**(**true**){

Article obj = dao.get(1);

obj.setHits(obj.getHits()+1);

**int** total = dao.update(obj);

**if**(total>0){

**break**;

}

}

select id,hits,version from t\_article where id = 1

update article set hits = #obj.hits,version=version+1 where id=#obj.id and version = #obj.version

1. 一个接口每次访问结果在上一次访问结果上+1，首次访问返回A，依次为B、C、…Z，ZA、ZB、…ZZ，ZZA….ZZZ。

**public** String request() {

**while** (**true**) {

String old = cnt.get();

String expect = **null**;

**if** (old == **null**) {

expect = "A";

} **else** {

**char** c = old.charAt(old.length() - 1);

**if** (c == 'Z') {

expect = old + 'A';

} **else** {

expect = String.*valueOf*(old.substring(0, old.length() - 1) + (**char**) (c + 1));

}

}

**if** (cnt.compareAndSet(old, expect)) {

**return** expect;

}

}

}

分布式：memcached cas/redis watch multi exec

1. class StuPk{int id,String school}

class Stu extends StuPk{String name, String age}

Map<StuPk,Stu> map = new HashMap<StuPk,Stu>();

map.put(new StuPk(1,”一中”), new Stu(1,”张三”,”男”,18))

map.put(new StuPk(2,”一中”), new Stu(1,”李四”,”男”,18))

map.get(new StuPk(1,”一中”)) 可以吗？不可以，hashcode和equals（）方法

1. List/Set/Map/StringBuilder/StringBuffer指定容量，避免扩容
2. 一个任务拆分为3个子任务由3个线程执行，子线程执行完成后由主线程对结果进行聚合，怎样实现？



FutureTask、join、countdownLatch

1. Spring aop实现原理（代理模式+责任链模式）
2. 3个业务方法m1/m2/m3，要求m1/m2/m3保证事务完整性，当异常时记录日志到数据库2个日志方法log1/log2，怎样实现？

@Transactional @Transactional

public void m(){ public void m(){

try{ m1();m2();m3();

m1();m2();m3(); }

} catch(Ex**c**eption ex){ public void exec(){

log(); try{

throw ex; m();

} }catch(Exception ex){log();}

} }

//@Transactional(propagation=Propagation.REQUIRES\_NEW)

public void log(){

log1();log2();

}

第一个，spring事务传播机制（在包含事务的方法外必须声明事务，不然事务不生效）

1. Synchronized/Lock/

synchronized是托管给JVM执行的，而lock是java写的控制锁的代码。在Java1.5中，synchronize是性能低效的。因为这是一个重量级操作，需要调用操作接口，导致有可能加锁消耗的系统时间比加锁以外的操作还多。相比之下使用Java提供的Lock对象，性能更高一些。但是到了Java1.6，发生了变化。synchronize在语义上很清晰，可以进行很多优化，有适应自旋，锁消除，锁粗化，轻量级锁，偏向锁等等。导致在Java1.6上synchronize的性能并不比Lock差。官方也表示，他们也更支持synchronize，在未来的版本中还有优化余地。

说到这里，还是想提一下这2中机制的具体区别。据我所知，synchronized原始采用的是CPU悲观锁机制，即线程获得的是独占锁。独占锁意味着其他线程只能依靠阻塞来等待线程释放锁。而在CPU转换线程阻塞时会引起线程上下文切换，当有很多线程竞争锁的时候，会引起CPU频繁的上下文切换导致效率很低。

而Lock用的是乐观锁方式。所谓乐观锁就是，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止。乐观锁实现的机制就是CAS操作（Compare and Swap）。我们可以进一步研究ReentrantLock的源代码，会发现其中比较重要的获得锁的一个方法是compareAndSetState。这里其实就是调用的CPU提供的特殊指令。

现代的CPU提供了指令，可以自动更新共享数据，而且能够检测到其他线程的干扰，而 compareAndSet() 就用这些代替了锁定。这个算法称作非阻塞算法，意思是一个线程的失败或者挂起不应该影响其他线程的失败或挂起的算法。