1. ReadWriteLock
   * + 1. 读写锁，为了提高性能同时也保证数据一致性，JDK中java.util.concurrent包里增加了ReadWriteLock

2.多个线程同时读一个资源类没有任何问题，所以为了满足并发量，读取共享资源应该可以同时进行。但是如果有一个线程想去写共享资源来，就不应该再有其它线程可以对该资源进行读或写小总结:

读-读能共存

读-写不能共存

写-写不能共存

范例:

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.HashMap;  import java.util.Map;  import java.util.concurrent.locks.ReadWriteLock;  import java.util.concurrent.locks.ReentrantReadWriteLock;  public class ReadWriteLockDemo {  public static void main(String[] args) {  Mycache mycache = new Mycache();  for (int i = 0; i < 5; i++) {  final int temp = i ;  new Thread(()->{  mycache.put(String.valueOf(temp),String.valueOf(temp));  }).start();  }  for (int i =0;i < 5 ;i++){  final int temp = i ;  new Thread(()->{  mycache.get(String.valueOf(temp));  }).start();  }  }  }  class Mycache{  private volatile Map<String,String> map = new HashMap<>();  private ReadWriteLock readWriteLock = new ReentrantReadWriteLock() ;  public void put (String key ,String value ){  readWriteLock.writeLock().lock();  try{  System.out.println("开始写入"+value);  map.put(key,value);  System.out.println("写入完成"+value);  }finally{  readWriteLock.writeLock().unlock();  }  }  public void get(String key){  readWriteLock.readLock().lock();  try{  }finally{  readWriteLock.readLock().unlock();  }  System.out.println("读取数据");  String s = map.get(key);  System.out.println("读取完成:"+s);  }  } |

读写分离，写时不能共享，而读可以共享。