多线程

Callable和Runnable

参考资料
Java并发编程：Callable、Future和FutureTask

ReentrantLock

BlockingQueue

BlockingQueue的特性：
当队列是空的时，从队列中获取或删除元素的操作将会被阻塞；
当队列是满时，往队列里添加元素的操作会被阻塞。

阻塞队列不接受空值，当你尝试向队列中添加空值的时候，它会抛出NullPointerException。
阻塞队列的实现都是线程安全的，所有的查询方法都是原子的并且使用了内部锁或者其他形式的并发控制。
BlockingQueue接口是java collections框架的一部分，它主要用于实现生产者-消费者问题。

final、static、static final修饰的字段赋值的区别：

static修饰的字段在类加载过程中的准备阶段被初始化为0或null等默认值，而后在初始化阶段（触发类构造器）才会被赋予代码中设定的值，如果没有设定值，那么它的值就为默认值；
final修饰的字段在运行时被初始化（可以直接赋值，也可以在实例构造器中赋值），一旦赋值便不可更改；
static final修饰的字段在Javac时生成ConstantValue属性，在类加载的准备阶段根据ConstantValue的值为该字段赋值，它没有默认值，必须显式地赋值，否则Javac时会报错。可以理解为在编译期即把结果放入了常量池中。

HashMap

参考资料

深入理解HashMap

Java虚拟机

参考资料

深入Java虚拟机系列

重要的类和代码

LRU Cache的实现

```
1  public class LRUCache {
2      private class Node{
3          Node prev;
4          Node next;
5          int key;
```

```
        int value;

        public Node(int key, int value) {
            this.key = key;
            this.value = value;
            this.prev = null;
            this.next = null;
        }
    }

    private int capacity;
    private HashMap<Integer, Node> hs = new HashMap<Integer, Node>();
    private Node head = new Node(-1, -1);
    private Node tail = new Node(-1, -1);

    public LRUCache(int capacity) {
        this.capacity = capacity;
        tail.prev = head;
        head.next = tail;
    }

    public int get(int key) {
        if( !hs.containsKey(key)) {
            return -1;
        }

        // remove current
        Node current = hs.get(key);
        current.prev.next = current.next;
        current.next.prev = current.prev;

        // move current to tail
        move_to_tail(current);

        return hs.get(key).value;
    }

    public void set(int key, int value) {
        if( get(key) != -1) {
            hs.get(key).value = value;
            return;
        }

        if (hs.size() == capacity) {
            hs.remove(head.next.key);
            head.next = head.next.next;
            head.next.prev = head;
        }

```

```
        Node insert = new Node(key, value);
        hs.put(key, insert);
        move_to_tail(insert);
    }

    private void move_to_tail(Node current) {
        current.prev = tail.prev;
        tail.prev = current;
        current.prev.next = current;
        current.next = tail;
    }
}
```