* 学习目标

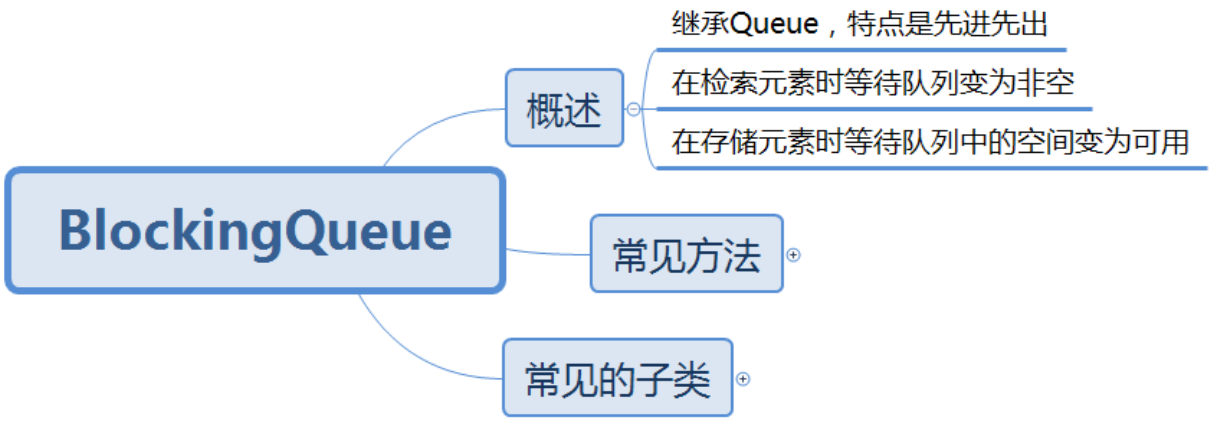
  * 能够掌握BlockingQueue相关阻塞队列

  * 能够掌握线程池

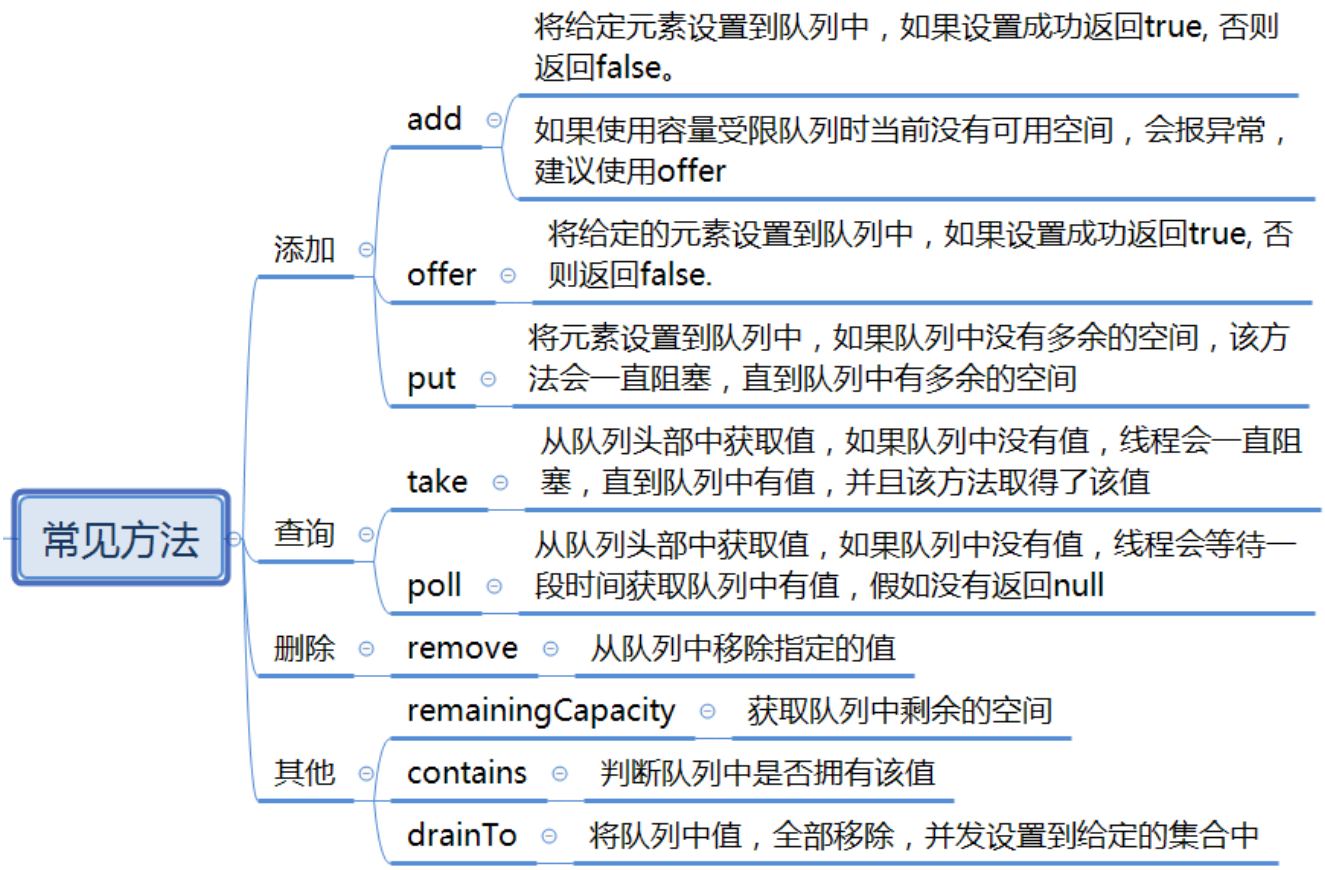
---

* 能够掌握BlockingQueue相关阻塞队列

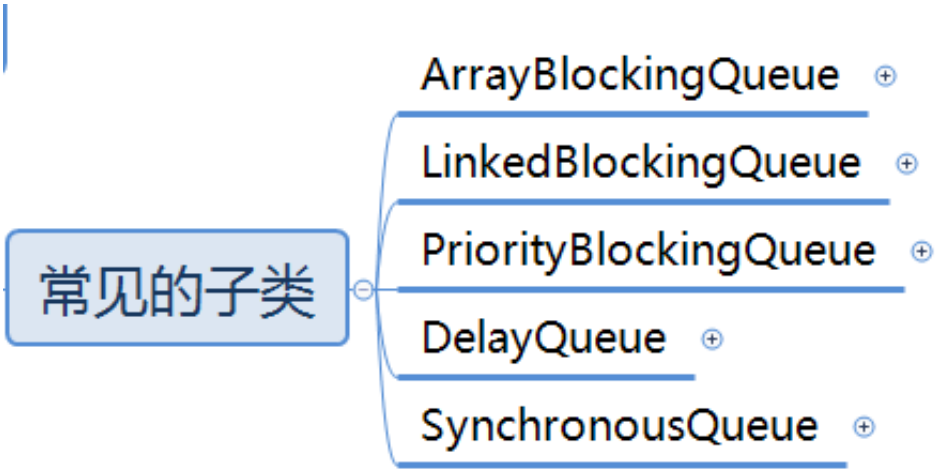
    * 概述



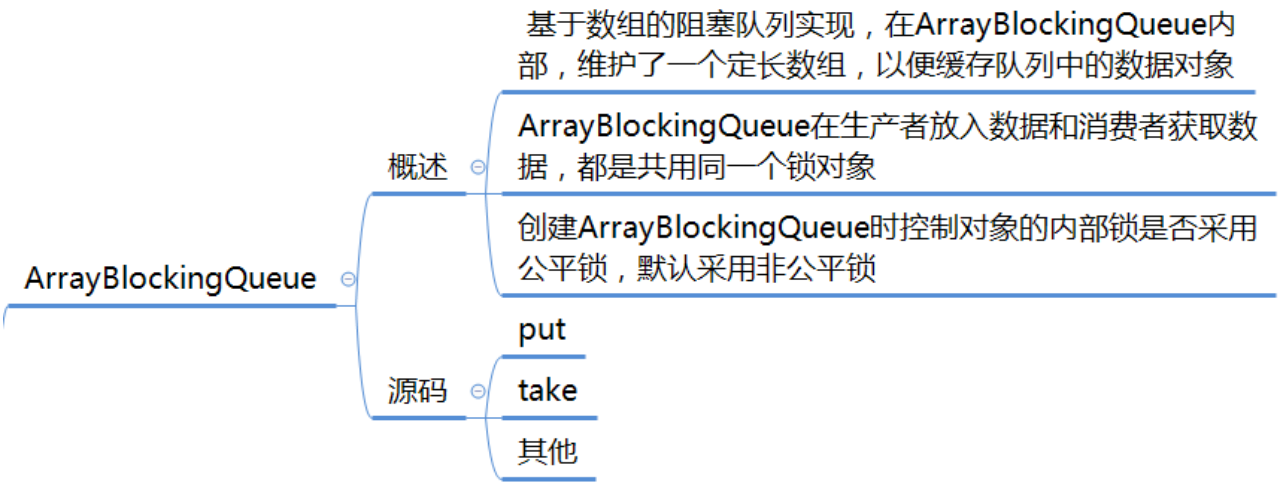
    * 常见方法



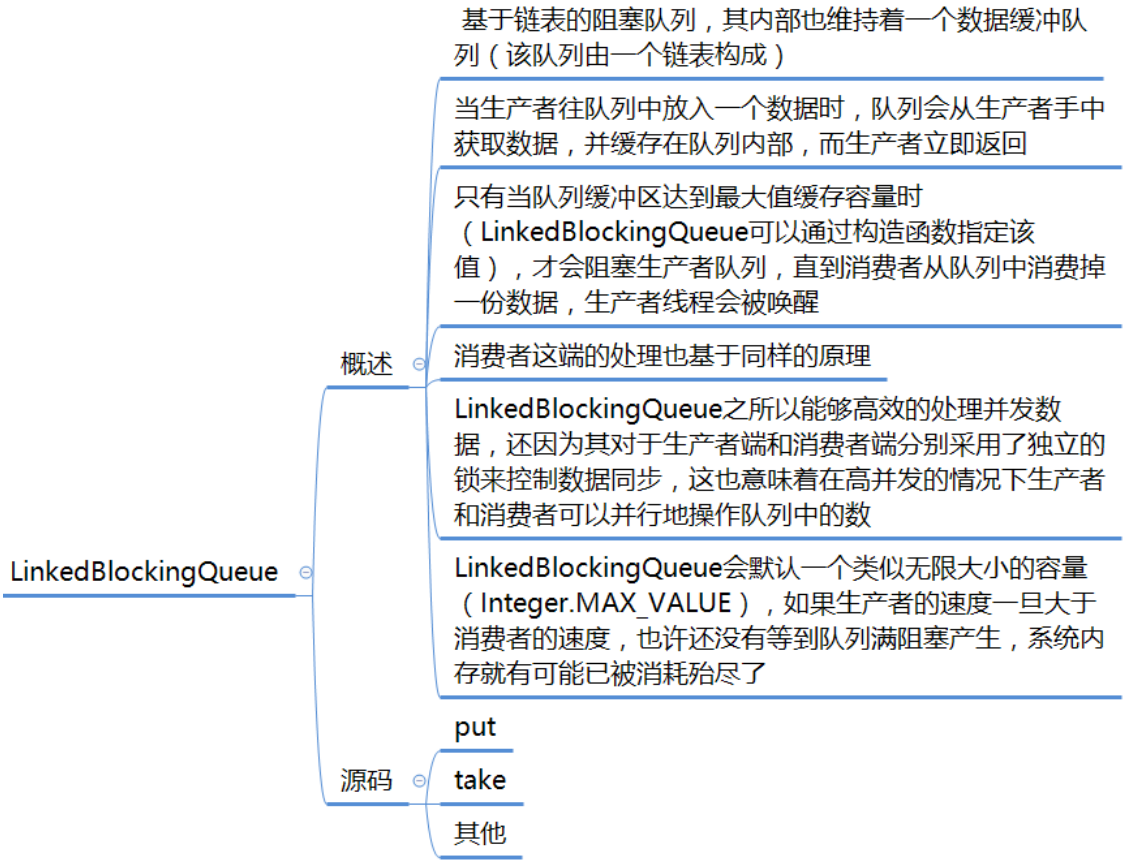
* 常见的子类



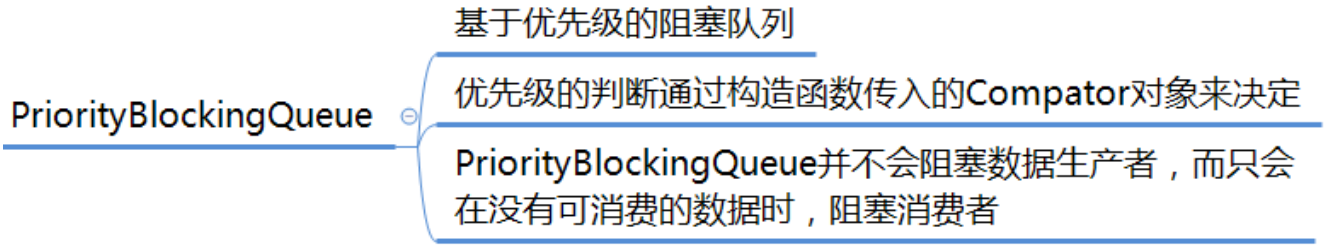
* ArrayBlockingQueue



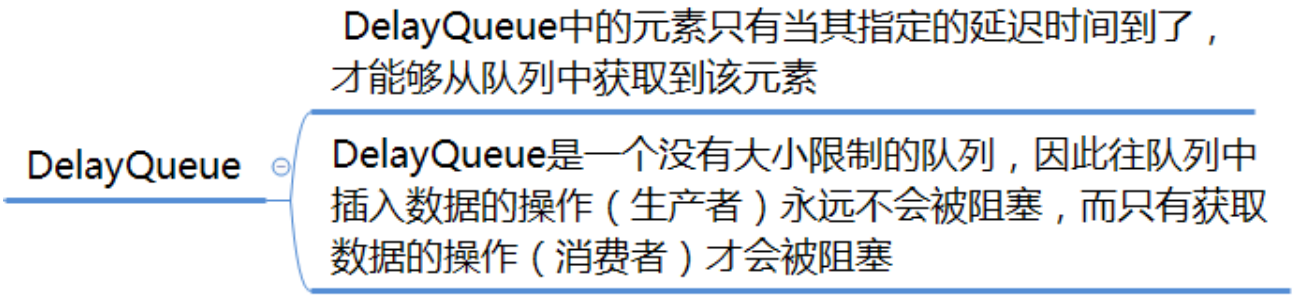
* LinkedBlockingQueue



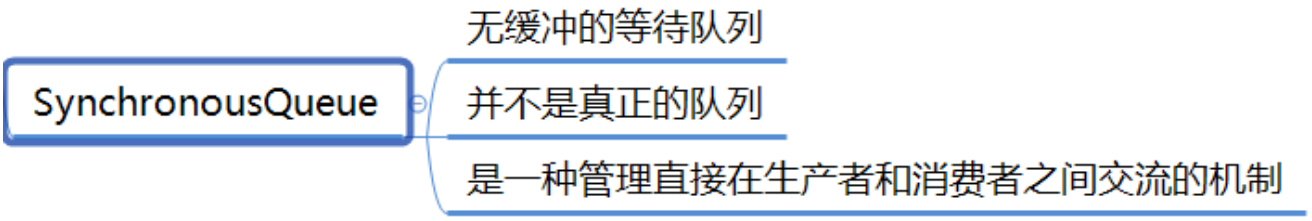
* PriorityBlockingQueue



* DelayQueue



* SynchronousQueue




```
* 案例：ArrayBlockingQueue
 * 生产者
public class Producer1 implements Runnable {
    private BlockingQueue mQueue;
    public Producer1(BlockingQueue queue){
        this.mQueue=queue;
    }
    @Override
    public void run() {
        // 一直在生产
        Random r=new Random();
        while (true){
                // 没有数据，就生产
            while (mQueue.size()<=0) {
                try {
                    Thread.sleep(2000);
                    int num = r.nextInt(50);
                    mQueue.put(num);
                    System.out.println(Thread.currentThread().getName() + "线程
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
```

```
    }
}
  * 消费者
public class Consumer1 implements Runnable {
    private BlockingQueue mQueue;
    public Consumer1(BlockingQueue queue){
       this.mQueue=queue;
    }
    @Override
    public void run() {
        while (true){
                while (mQueue.size() > 0) {
                    try {
                        Thread.sleep(1000);
                        System.err.println(Thread.currentThread().getName()+"线
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
        }
    }
}
 * Main
public class Main {
    public static void main(String[] args)  {
        BlockingQueue blockingQueue=new ArrayBlockingQueue(16);
        for (int i = 0; i < 5; i++) {
            // 生产者
            new Thread(new Producer1(blockingQueue),"生产者"+i){}.start();
            // 消费者
            new Thread(new Consumer1(blockingQueue),"消费者"+i){}.start();
        }
    }

}

* 案例二：LinkedBlockedQueue
  * ArrayBlockingQueue 替换成LinkedBlockedQueue

* 案例三：PriorityBlockingQueue
```

```java
@Data
@AllArgsConstructor
public class Task implements Comparable<Task> {
    private int id;
    private String name;
    @Override
    public int compareTo(Task o) {
        return this.getId()-o.id ;
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        BlockingQueue<Task> blockingQueue=new PriorityBlockingQueue<Task> (16);
        Task task1=new Task(1,"xiaohei");
        Task task3=new Task(3,"xiaobai");
        Task task2=new Task(2,"xiaoming");
        blockingQueue.put(task1);
        blockingQueue.put(task2);
        blockingQueue.put(task3);
        for (Task task:blockingQueue){
            System.out.println(task);
        }
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(3000);
                    blockingQueue.put(new Task(5,"xiaohong"));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();
        List<Task> tasks=new ArrayList();
        blockingQueue.drainTo(tasks);
        Task task = blockingQueue.take();
        System.out.println("task = " + task);
        System.out.println("main over");

```

```
105
106     }
107 }
108
109 * 案例网吧上网 :DelayQueue
110  * 实现 Delayed接口
111 @Data
112 public class Customer implements Delayed {
113     private int id;
114     private String name;
115     private long endTime;
116
117     @Override
118     public long getDelay(TimeUnit unit) {
119         return this.endTime-System.currentTimeMillis();
120     }
121
122     public Customer(int id, String name, long endTime) {
123         this.id = id;
124         this.name = name;
125         // 加上当前时间
126         this.endTime = endTime+System.currentTimeMillis();
127     }
128
129     @Override
130     public int compareTo(Delayed o) {
131         return (int) (this.endTime-((Customer) o).endTime);
132     }
133 }
134  * DownTask
135 public class DownTask implements Runnable {
136     private BlockingQueue<Customer> mQueue;
137     public DownTask(BlockingQueue<Customer> blockingQueue){
138             this.mQueue=blockingQueue;
139     }
140     @Override
141     public void run() {
142         while (true) {
143             try {
144                 Customer customer = mQueue.take();
```

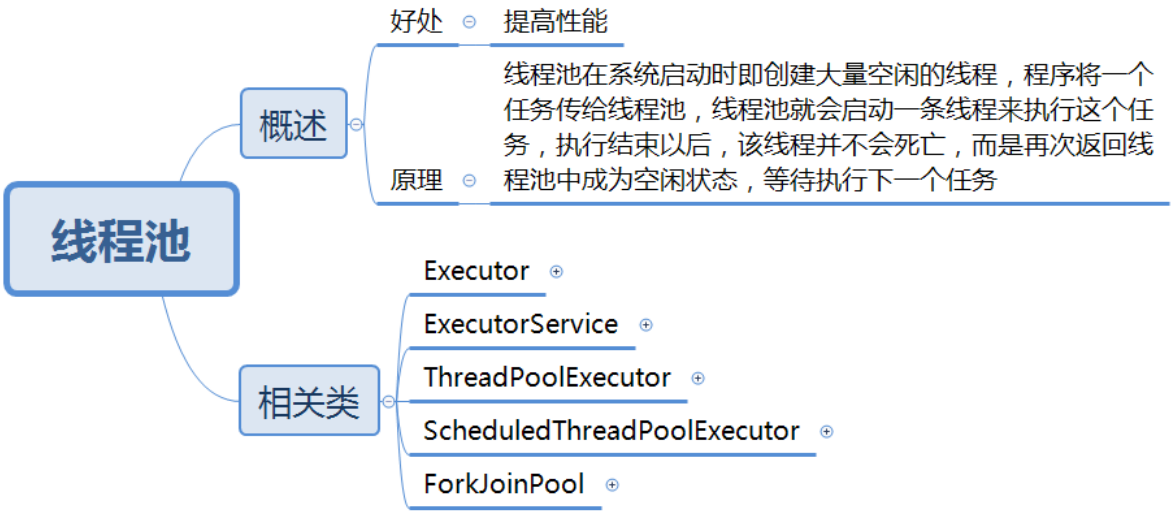
```
                System.out.println("编号: " + customer.getId() + " 姓名: " + cus
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

* Main
public class Main {
    public static void main(String[] args) {
        BlockingQueue<Customer> queue=new DelayQueue<Customer>();
        Customer c1=new Customer(1001,"刘备",1000*5);
        Customer c2=new Customer(1002,"关羽",1000*10);
        Customer c3=new Customer(1003,"张飞",1000*20);
        queue.add(c1);
        queue.add(c2);
        queue.add(c3);
        new Thread(new DownTask(queue)).start();
        SimpleDateFormat sdf =new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        while (true){
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(sdf.format(new Date()));
        }
    }
}

* 案例：SynchronousQueue
  * 生产者
public class Producer1 implements Runnable {
    private BlockingQueue mQueue;
    public Producer1(BlockingQueue queue){
        this.mQueue=queue;
    }
    @Override
    public void run() {
```

```
        // 一直在生产
        Random r=new Random();
        while (true){
                try {
                    Thread.sleep(2000);
                    int num = r.nextInt(50);
                    mQueue.put(num);
                    System.out.println(Thread.currentThread().getName() + "线程
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

        }
    }
}

  * 消费者
public class Consumer1 implements Runnable {
    private BlockingQueue mQueue;
    public Consumer1(BlockingQueue queue){
       this.mQueue=queue;
    }
    @Override
    public void run() {
        while (true){
                    try {
                        Thread.sleep(1000);
                        System.err.println(Thread.currentThread().getName()+"线
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
        }
    }
}

 * Main
public class Main {
    public static void main(String[] args)  {
        BlockingQueue blockingQueue=new SynchronousQueue();
            // 生产者
```

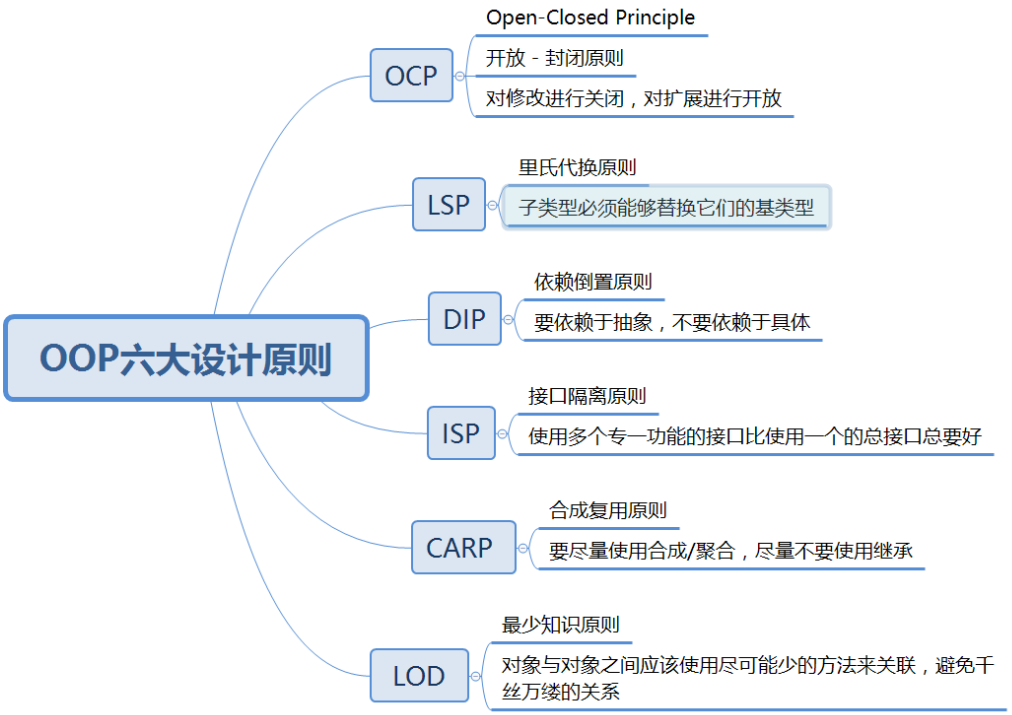
```
225         new Thread(new Producer1(blockingQueue),"生产者"){}.start();
226             // 消费者
227         new Thread(new Consumer1(blockingQueue),"消费者"){}.start();
228     }
229 }
230
```

* 能够掌握线程池

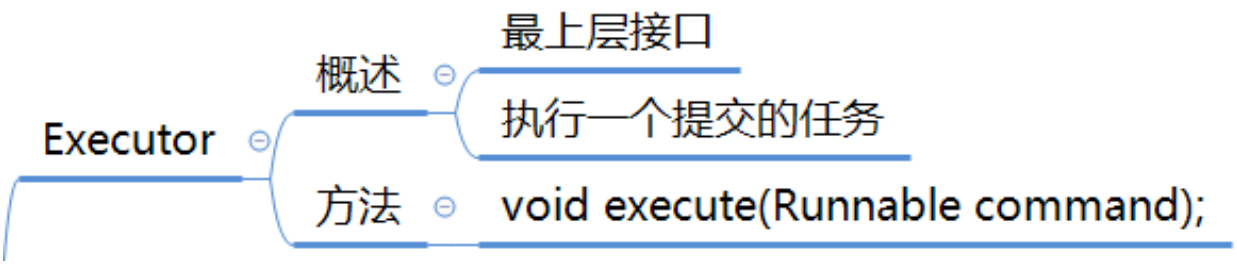
  * 线程池概述



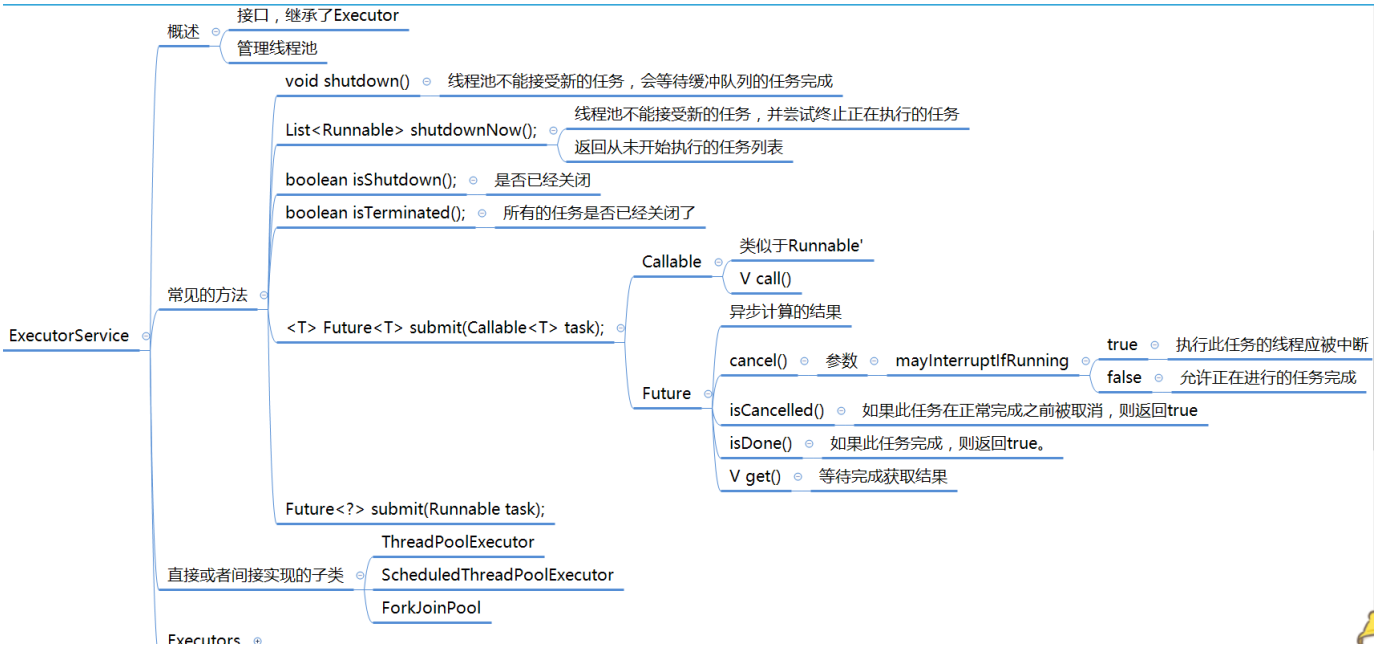
* 面向对象六大设计原则



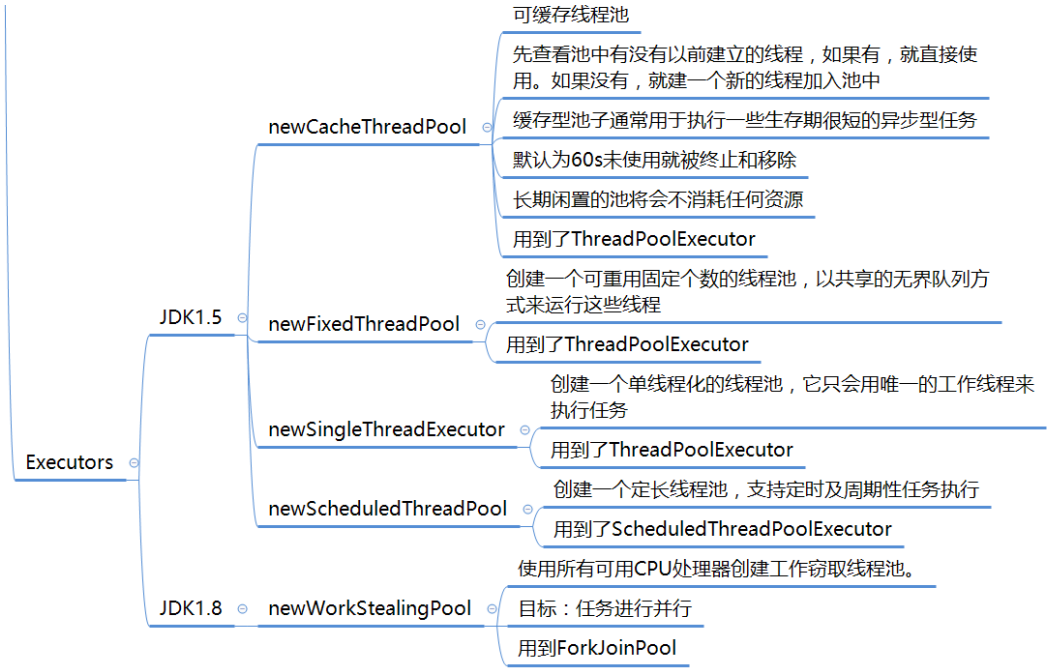
* Executor



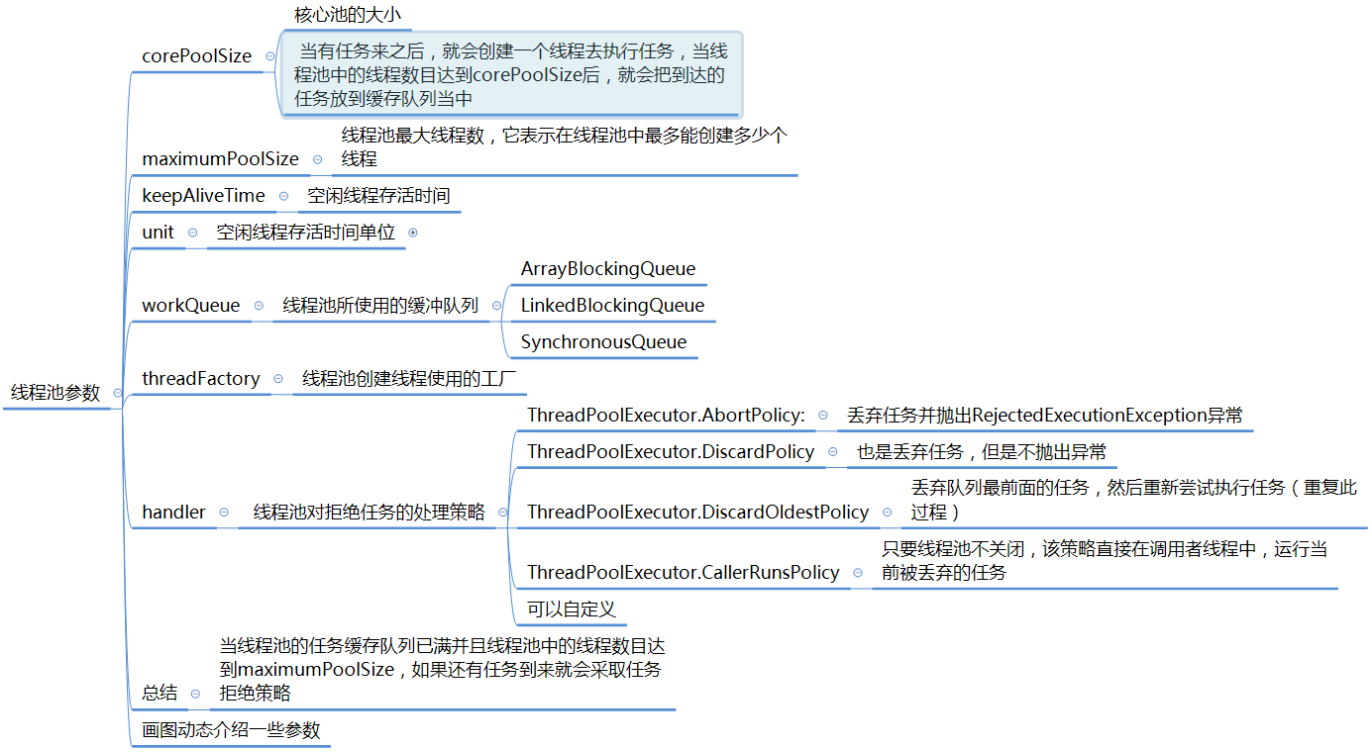
* ExecutorService



* Executors



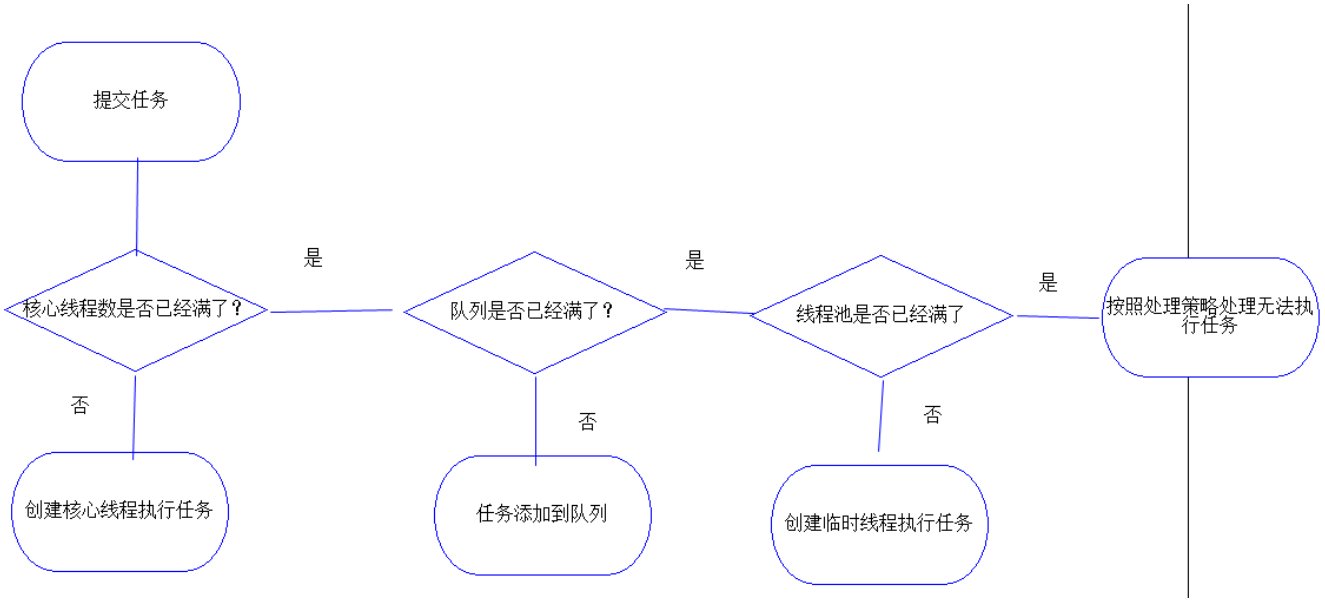
* ThreadPoolExecutor



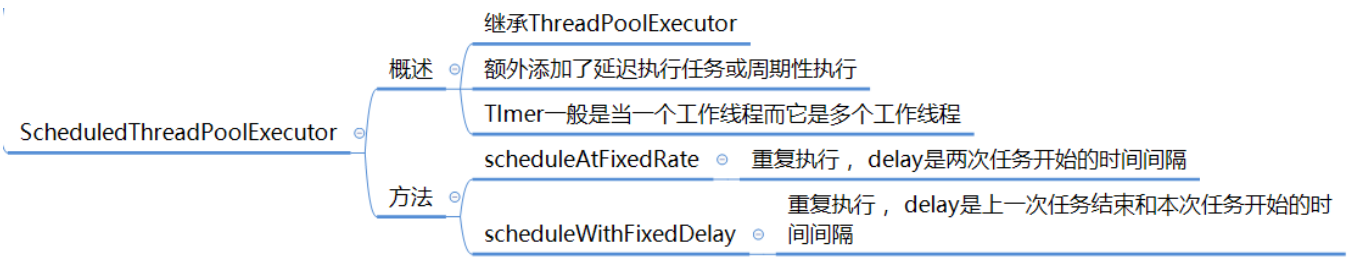
* 线程池执行任务逻辑和线程池参数的关系

  * 西游记的故事

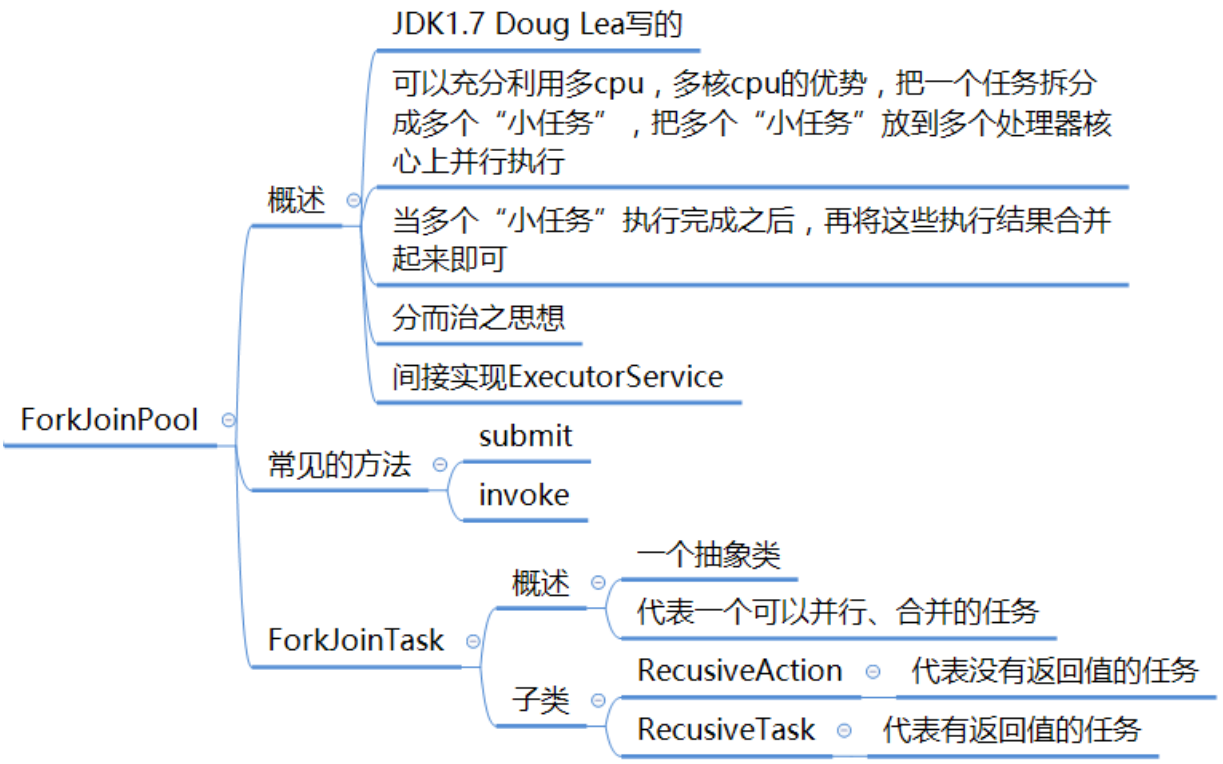
* 流程图



* ScheduledThreadPoolExecutor



* ForkJoinPool




```
案例一：Executors
public class Main {
    public static void main(String[] args) {
        // 阿里建议手动创建线程池，后面讲解
        ExecutorService executorService = Executors.newCachedThreadPool();
        //Executors.newSingleThreadExecutor();
        //Executors.newFixedThreadPool(5);
        for (int i = 0; i < 100; i++) {
            int temp = i;
            executorService.execute(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(Thread.currentThread().getName()+"线程被
                }
            });
        }

    }
```

```
}
 * 查看效果体会
* 案例二：ScheduledThreadPoolExecutor：定时的效果
          Timer
public class Main {
    public static void main(String[] args) {
        // 阿里建议手动创建线程池，后面讲解
        ScheduledThreadPoolExecutor schedule =
                (ScheduledThreadPoolExecutor) Executors.newScheduledThreadPool(
        // 1 秒后执行此任务
        schedule.schedule(new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName()+"线程被执行
            }
        },1,TimeUnit.SECONDS);
        // 是固定的频率来执行某项计划，它不受计划执行时间的影响
        schedule.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.err.println(Thread.currentThread().getName()+"线程被执行
            }
        },1000,5000, TimeUnit.MILLISECONDS);
        //无论某个任务执行多长时间，等执行完了，我再延迟指定的时间
        schedule.scheduleWithFixedDelay(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName()+"线程被执行
            }
        },1000,5000, TimeUnit.MILLISECONDS);
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        // 阿里建议是ScheduledThreadPoolExecutor代替Timer
        Timer timer=new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.err.println(Thread.currentThread().getName()+"线程被执行
            }
        },1000,2000);
    }
}

* 案例三：ExecutorService 常见的方法
public class Main {
    public static void main(String[] args) throws Exception {
        ExecutorService pool = Executors.newFixedThreadPool(5);
        for (int i = 0; i < 10; i++) {
            Future<String> future = pool.submit(new Callable<String>() {
                @Override
                public String call() throws Exception {
                    Thread.sleep(3000);
                    System.out.println(Thread.currentThread().getName()+"线程被
                    return "success";
                }
            });
            System.out.println(future.get());
        }
        pool.shutdown();
//        pool.shutdownNow();
    }
}

* 案例四：ForkJoinPool
public class Main {
    public static void main(String[] args) throws Exception {
        ForkJoinPool pool=new ForkJoinPool();
```

```
        //创建实例，并执行分割任务
        for (int i = 0; i < 10; i++) {
            ForkJoinTask<Void> submit = pool.submit(new RecursiveAction() {
                @Override
                protected void compute() {
                    System.out.println(Thread.currentThread().getName()+"线程被
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
        //线程阻塞，等待所有任务完成
        pool.awaitTermination(10,TimeUnit.SECONDS);
        pool.shutdown();
    }
}

* 案例五：
public class Main {
    public static void main(String[] args) throws Exception {
        int corePoolSize=5;
        int maximumPoolSize=8;
        long keepAliveTime=5;
        TimeUnit unit= TimeUnit.SECONDS;
        BlockingQueue<Runnable> workQueue=new LinkedBlockingDeque<>(10);
        ThreadFactory threadFactory=Executors.defaultThreadFactory();
        RejectedExecutionHandler handler=new ThreadPoolExecutor.AbortPolicy();
        ThreadPoolExecutor pool=new ThreadPoolExecutor(corePoolSize,maximumPool
               keepAliveTime,unit,workQueue,threadFactory,handler);
        // 5 测试
        // 6 测试
        //16 测试
        // 19 测试
        // 更换拒绝策略，自定义策略
        for (int i = 0; i <5 ; i++) {
            int temp=i;
            pool.submit(new Callable<String>() {
```

```
                @Override
                public String call() throws Exception {
                    Thread.sleep(1000);
                    System.out.println(Thread.currentThread().getName()+"执行了
                    return "Success";
                }
            });
        }
        pool.shutdown();
    }
}

* 案例六：自定义ThreadPool
public class ThreadPoolUtils {
    private static int corePoolSize;
    private static int maximumPoolSize;
    private static long keepAliveTime;
    private static String capcity;
    private static RejectedExecutionHandler handler;
    private static BlockingQueue<Runnable> workQueue;
    private static TimeUnit unit= TimeUnit.MILLISECONDS;
    private static ThreadFactory threadFactory= Executors.defaultThreadFactory(
    private static ThreadPoolExecutor pool;
    private ThreadPoolUtils(){

    }
    static {
        Properties prop=new Properties();
        try {
            prop.load(ThreadPoolUtils.class.getClassLoader().getResourceAsStrea
        } catch (IOException e) {
            e.printStackTrace();
        }
        corePoolSize = Integer.parseInt(prop.getProperty("corePoolSize", "5"));
        maximumPoolSize= Integer.parseInt(prop.getProperty("maximumPoolSize","8
        keepAliveTime =Long.parseLong(prop.getProperty("keepAliveTime", "1000")
        capcity = prop.getProperty("capcity","0");
        if(capcity.equals("0")){
            workQueue=new LinkedBlockingDeque<>();
        }else{
```

```
            workQueue=new LinkedBlockingDeque<>(Integer.parseInt(capcity));
        }
        String handler1 = prop.getProperty("handler","1");
        switch (handler1){
            case "1":
                handler=new ThreadPoolExecutor.AbortPolicy();
                break;
            case "2":
                handler=new ThreadPoolExecutor.DiscardPolicy();
                break;
            case "3":
                handler=new ThreadPoolExecutor.DiscardOldestPolicy();
                break;
            case "4":
                handler=new ThreadPoolExecutor.CallerRunsPolicy();
                break;
             default:
                 handler=new ThreadPoolExecutor.AbortPolicy();
                 break;
        }
        pool=new ThreadPoolExecutor(corePoolSize,maximumPoolSize,keepAliveTime,
    }

    public static ThreadPoolExecutor getPool(){
        return pool;
    }
}
* 配置文件
corePoolSize=5
maximumPoolSize=8
keepAliveTime=1000
capcity=10
-- 1:AbortPolicy|DiscardPolicy|DiscardOldestPolicy|CallerRunsPolicy
handler=1

* 测试代码
public class Main {
    public static void main(String[] args) throws Exception {
        // 5 测试
        // 6 测试
```

```
        //16 测试
        // 19 测试
        // 更换拒绝策略，自定义策略
        for (int i = 0; i <19 ; i++) {
            int temp=i;
            ThreadPoolUtils.getPool().submit(new Callable<String>() {
                @Override
                public String call() throws Exception {
                    Thread.sleep(1000);
                    System.out.println(Thread.currentThread().getName()+"执行了
                    return "Success";
                }
            });
        }
        ThreadPoolUtils.getPool().shutdown();
    }
}


```