线程池

池化技术 池化思想

优势：

- 提高线程的利用率
- 提高响应速度
- 便于统一管理线程对象
- 可控制最大并发数

线程池的具体设计思想

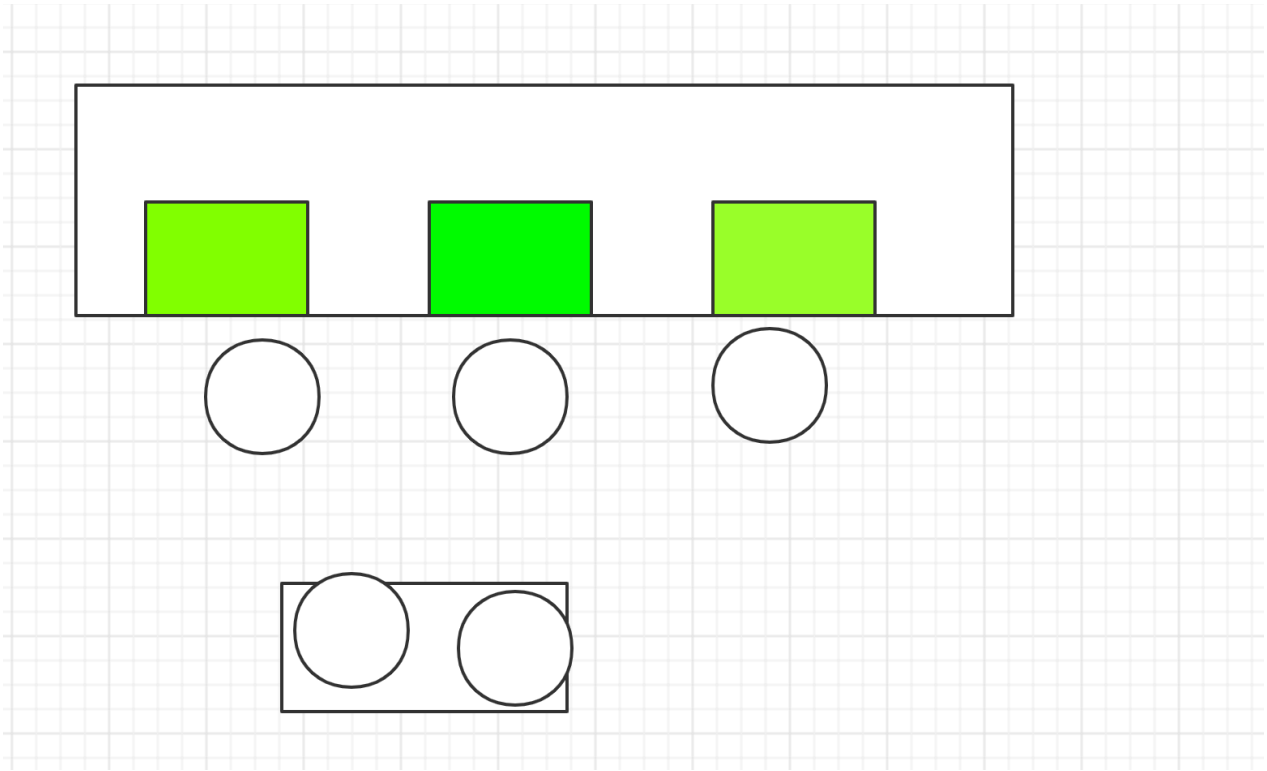
- 核心池的大小
- 线程池的最大容量
- 等待队列
- 拒绝策略

线程池启动的时候会按照核心池的数来创建初始化的线程对象 2 个。

开始分配任务，如果同时来了多个任务， 2 个线程对象都被占用了，第 3 个以及之后的任务进入等待队列，当前有线程完成任务恢复空闲状态的时候，等待队列中的任务获取线程对象。

如果等待队列也占满了，又有新的任务进来，需要去协调，让线程池再创建新的线程对象，但是线程池不可能无限去创建线程对象，一定会有一个最大上限，就是线程池的最大容量。

如果线程池已经达到了最大上限，并且等待队列也占满了，此时如果有新的任务进来，只能选择拒绝，并且需要根据拒绝策略来选择对应的方案。



ThreadPoolExecutor

直接实例化 ThreadPoolExecutor ，实现定制化的线程池，而不推荐使用 Executors 提供的封装好的方法，因为这种方式代码不够灵活，无法实现定制化。

ThreadPoolExecutor 核心参数一共有 7 个

```
corePoolSize：核心池的大小
maximumPoolSize：线程池的最大容量
keepAliveTime：线程存活时间（在没有任务可执行的情况下），必须是线程池中的数量大于
corePoolSize，才会生效
TimeUnit：存活时间单位
BlockingQueue：等待队列，存储等待执行的任务
ThreadFactory：线程工厂，用来创建线程对象
RejectedExecutionHandler：拒绝策略
1、AbortPolicy：直接抛出异常
2、DiscardPolicy：放弃任务，不抛出异常
3、DiscardOldestPolicy：尝试与等待队列中最前面的任务去争夺，不抛出异常
4、CallerRunsPolicy：谁调用谁处理
```

单例 1

```
 */
@NotNull public static ExecutorService newSingleThreadExecutor() {
    return new FinalizableDelegatedExecutorService
        (new ThreadPoolExecutor( corePoolSize: 1,  maximumPoolSize: 1,
                                keepAliveTime: 0L, TimeUnit.MILLISECONDS,
                                new LinkedBlockingQueue<Runnable>()));
}
```

固定 5

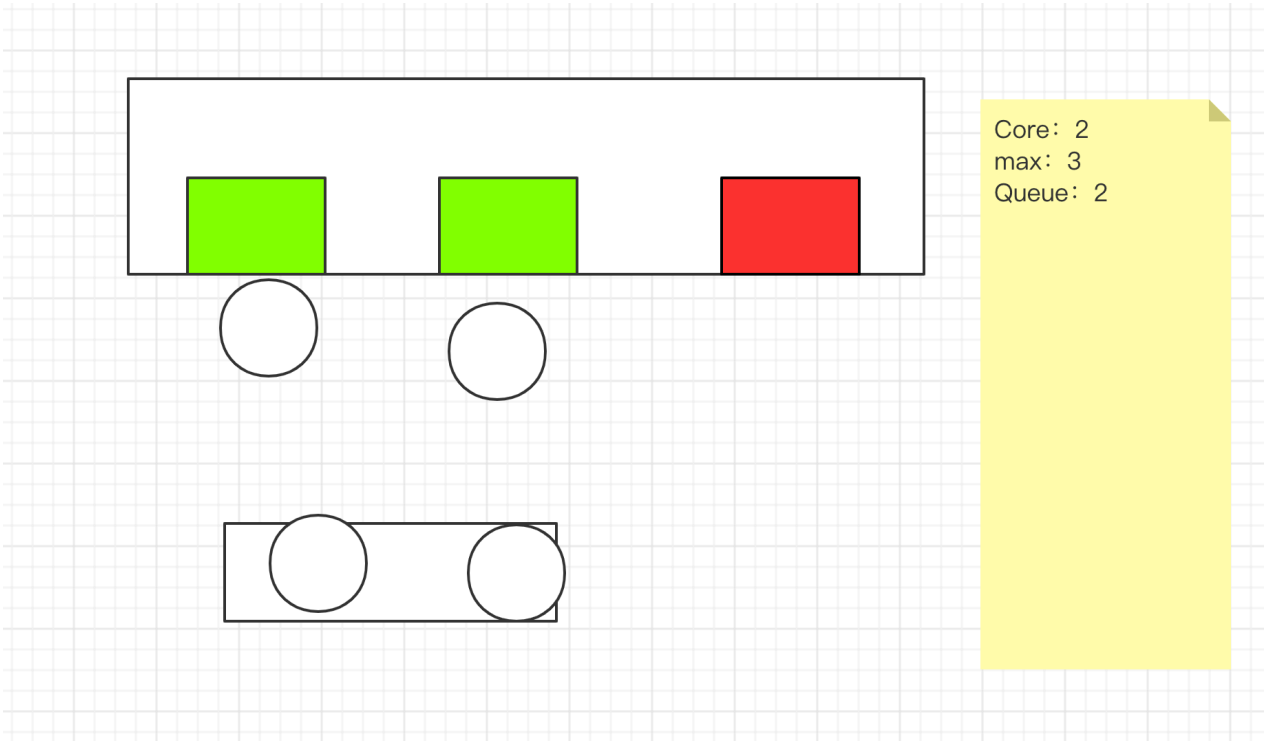
```
public static ExecutorService newFixedThreadPool(int nThreads) {
    return new ThreadPoolExecutor(nThreads, nThreads,
                                  keepAliveTime: 0L, TimeUnit.MILLISECONDS,
                                  new LinkedBlockingQueue<Runnable>());
}
```

缓存

```
@NotNull public static ExecutorService newCachedThreadPool() {
    return new ThreadPoolExecutor( corePoolSize: 0, Integer.MAX_VALUE,
                                   keepAliveTime: 60L, TimeUnit.SECONDS,
                                   new SynchronousQueue<Runnable>());
}
```

```
package com.southwind.demo2;

import java.util.concurrent.*;

public class Test {
```

```
    public static void main(String[] args) {
        ExecutorService executorService = null;
        try {
            /**
             * 自己写7大参数，完全定制化
             */
            executorService = new ThreadPoolExecutor(
                    2,
                    3,
                    1L,
                    TimeUnit.SECONDS,
                    new ArrayBlockingQueue<>(2),
                    Executors.defaultThreadFactory(),
                    new ThreadPoolExecutor.AbortPolicy()
            );

            for (int i = 0; i < 6; i++) {
                executorService.execute(()->{
                    try {
                        TimeUnit.MILLISECONDS.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(Thread.currentThread().getName()+"===>办
理业务");
                });
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            executorService.shutdown();
        }
    }
}
```



new ThreadPoolExecutor.AbortPolicy()

```
/Library/Java/JavaVirtualMachines/jdk-10.0.1.jdk/Conte
java.util.concurrent.RejectedExecutionException: Task
    at java.base/java.util.concurrent.ThreadPoolExecut
    at java.base/java.util.concurrent.ThreadPoolExecut
    at java.base/java.util.concurrent.ThreadPoolExecut
    at com.southwind.demo2.Test.main(Test.java:23)
pool-1-thread-3===>办理业务
```

new ThreadPoolExecutor.CallersunsPolicy()

```
Test (1)
/Library/Java/JavaVirtualMachines/jdk-10.0.1
pool-1-thread-1===>办理业务
pool-1-thread-2===>办理业务
pool-1-thread-3===>办理业务
main===>办理业务
pool-1-thread-2===>办理业务
pool-1-thread-1===>办理业务
```

new ThreadPoolExecutor.DiscardOldestPolicy()

new ThreadPoolExecutor.DiscardPolicy()

不会抛出异常

线程池 3 大考点：

1、Executors 工具类的 3 种实现

```
ExecutorService executorService = Executors.newSingleThreadExecutor();
ExecutorService executorService = Executors.newFixedThreadPool(5);
ExecutorService executorService = Executors.newCachedThreadPool();
```

2、7 个参数

```
corePoolSize：核心池的大小
maximumPoolSize：线程池的最大容量
keepAliveTime：线程存活时间（在没有任务可执行的情况下），必须是线程池中的数量大于
corePoolSize，才会生效
TimeUnit：存活时间单位
BlockingQueue：等待队列，存储等待执行的任务
ThreadFactory：线程工厂，用来创建线程对象
RejectedExecutionHandler：拒绝策略
```

3、4 种拒绝策略

```
1、AbortPolicy：直接抛出异常
2、DiscardPolicy：放弃任务，不抛出异常
3、DiscardOldestPolicy：尝试与等待队列中最前面的任务去争夺，不抛出异常
4、CallerRunsPolicy：谁调用谁处理
```