20240325腾讯IDC一面

项目

八股

java实现多线程的方式，runnable和callable有什么差异

https://segmentfault.com/a/1190000022878543

(1) Callable规定（重写）的方法是call()，Runnable规定（重写）的方法是run()。

(2) Callable的任务执行后可返回值，而Runnable的任务是不能返回值的。

(3) call方法可以抛出异常，run方法不可以。

(4) 运行Callable任务可以拿到一个Future对象，表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。通过Future对象可以了解任务执行情况，可取消任务的执行，还可获取执行结果。

java线程池有几种

https://javaguide.cn/java/concurrent/java-thread-pool-summary.html#几种常见的内置线程池

https://www.cnblogs.com/pcheng/p/13540619.html

FixedThreadPool

`FixedThreadPool` 被称为可重用固定线程数的线程池。`FixedThreadPool` 的 `corePoolSize` 和 `maximumPoolSize` 都被设置为 `nThreads`，这个 `nThreads` 参数是我们使用的时候自己传递的。

即使 `maximumPoolSize` 的值比 `corePoolSize` 大，也至多只会创建 `corePoolSize` 个线程。这是因为`FixedThreadPool` 使用的是容量为 `Integer.MAX_VALUE` 的 `LinkedBlockingQueue`（无界队列），队列永远不会被放满。

运行流程：

1. 如果当前运行的线程数小于 `corePoolSize`， 如果再来新任务的话，就创建新的线程来执行任务；
2. 当前运行的线程数等于 `corePoolSize` 后， 如果再来新任务的话，会将任务加入 `LinkedBlockingQueue`；
3. 线程池中的线程执行完 手头的任务后，会在循环中反复从 `LinkedBlockingQueue` 中获取任务来执行；

`FixedThreadPool` 使用无界队列 `LinkedBlockingQueue`（队列的容量为 Integer.MAX_VALUE）作为线程池的工作队列会对线程池带来如下影响：

1. 当线程池中的线程数达到 `corePoolSize` 后，新任务将在无界队列中等待，因此线程池中的线程数不会超过 `corePoolSize`；
2. 由于使用无界队列时 `maximumPoolSize` 将是一个无效参数，因为不可能存在任务队列满的情况。所以，通过创建 `FixedThreadPool`的源码可以看出创建的 `FixedThreadPool` 的 `corePoolSize` 和 `maximumPoolSize` 被设置为同一个值。
3. 由于 1 和 2，使用无界队列时 `keepAliveTime` 将是一个无效参数；
4. 运行中的 `FixedThreadPool`（未执行 `shutdown()`或 `shutdownNow()`）不会拒绝任务，在任务比较多的时候会导致 OOM（内存溢出）。

```
ExecutorService executorService = Executors.newFixedThreadPool(3
        for (int i = 0; i < 10; i++) {
            final int index = i;
            executorService.execute(() -> {
                // 获取线程名称,默认格式:pool-1-thread-1
                System.out.println(Thread.currentThread().getNam
                // 等待2秒
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            });
        }
// 因为线程池大小是固定的，这里设置的是3个线程，所以线程名只有3个。因为线程
```

SingleThreadExecutor

`SingleThreadExecutor` 是只有一个线程的线程池。
`SingleThreadExecutor` 的 `corePoolSize` 和 `maximumPoolSize` 都被设置为 1，其他参数和 `FixedThreadPool` 相同。

运行流程如下：

1. 如果当前运行的线程数少于 `corePoolSize`，则创建一个新的线程执行任务；
2. 当前线程池中有一个运行的线程后，将任务加入 `LinkedBlockingQueue`
3. 线程执行完当前的任务后，会在循环中反复从`LinkedBlockingQueue` 中获取任务来执行；

`SingleThreadExecutor` 和 `FixedThreadPool` 一样，使用的都是容量为 `Integer.MAX_VALUE` 的 `LinkedBlockingQueue`（无界队列）作为线程池的工作队列。`SingleThreadExecutor` 使用无界队列作为线程池的工作队列会对线程池带来的影响与 `FixedThreadPool` 相同。说简单点，就是可能会导致 OOM。

```
ExecutorService executorService = Executors.newSingleThreadExec
        for (int i = 0; i < 10; i++) {
            final int index = i;
            executorService.execute(() -> {
                // 获取线程名称,默认格式:pool-1-thread-1
                System.out.println(Thread.currentThread().getNar
                // 等待2秒
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            });
        }
// 因为只有一个线程，所以线程名均相同，且是每隔2秒按顺序输出的。
```

CachedThreadPool

`CachedThreadPool` 是一个会根据需要创建新线程的线程池。`CachedThreadPool` 的 `corePoolSize` 被设置为空（0），`maximumPoolSize`被设置为 `Integer.MAX.VALUE`，即它是无界的。这也就意味着如果主线程提交任务的速度高于 `maximumPool` 中线程处理任务的速度时，`CachedThreadPool` 会不断创建新的线程。极端情况下，这样会导致耗尽 cpu 和内存资源。

`CachedThreadPool` 使用的是同步队列 `SynchronousQueue`, 允许创建的线程数量为 `Integer.MAX_VALUE` ，可能会创建大量线程，从而导致 OOM。

```
ExecutorService executorService = Executors.newCachedThreadPool(
        for (int i = 0; i < 10; i++) {
            final int index = i;
            executorService.execute(() -> {
                // 获取线程名称,默认格式:pool-1-thread-1
                System.out.println(Thread.currentThread().getNar
                // 等待2秒
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
```

```
                }
            });
        }

// 因为初始线程池没有线程，而线程不足会不断新建线程，所以输出的线程名都是不
```

ScheduledThreadPool

`ScheduledThreadPool` 用来在给定的延迟后运行任务或者定期执行任务。`ScheduledThreadPool` 是通过 `ScheduledThreadPoolExecutor` 创建的，使用的 `DelayedWorkQueue`（延迟阻塞队列）作为线程池的任务队列。

`DelayedWorkQueue` 的内部元素并不是按照放入的时间排序，而是会按照延迟的时间长短对任务进行排序，内部采用的是“堆”的数据结构，可以保证每次出队的任务都是当前队列中执行时间最靠前的。`DelayedWorkQueue` 添加元素满了之后会自动扩容原来容量的 1/2，即永远不会阻塞，最大扩容可达 `Integer.MAX_VALUE`，所以最多只能创建核心线程数的线程。

io流，读取文件内容的api

https://blog.csdn.net/QQ578473688/article/details/70198060

https://blog.csdn.net/QQ578473688/article/details/70198060

innodb引擎优点

mysql相关的锁

spring、springmvc、springboot和spring cloud之间的关系

Spring是一个一站式的轻量级的java开发框架，核心是控制反转（IOC）和面向切面（AOP），针对于开发的WEB层(springMvc)、业务层(Ioc)、持久层(jdbcTemplate)等都提供了多种配置解决方案；

SpringMVC是Spring基础之上的一个MVC框架，主要处理web开发的路径映射和视图渲染，属于Spring框架中WEB层开发的一部分。

SpringBoot使用了默认大于配置的理念，集成了快速开发的Spring多个插件，同时自动过滤不需要配置的多余的插件，简化了项目的开发配置流程，一定程度上取消xml配置，是一套快速配置开发的脚手架，能快速开发单个微服务；

SpringCloud大部分的功能插件都是基于SpringBoot去实现的，SpringCloud关注于全局的微服务整合和管理，将多个SpringBoot单体微服务进行整合以及管理；
SpringCloud依赖于SpringBoot开发，而SpringBoot可以独立开发；

spring cloud由什么组成

linux查看进程使用资源的情况的命令

1. top：top是一个实时监控系统资源使用情况的命令。在终端输入top命令，然后按下“p”键，输入进程ID，可以查看指定进程的资源使用情况。使用“q”键可以退出top命令。

2. ps：ps命令用于列出当前运行进程的信息。在终端输入“ps -p 进程ID”，可以查看指定进程的信息，包括进程ID、进程状态、内存使用情况等。
3. htop：htop是top命令的增强版，它可以以交互式的方式显示系统资源使用情况。在终端输入htop命令，然后按下“F6”键，输入进程ID，可以查看指定进程的资源使用情况。

linux从一台主机查看另一台主机某个端口的是否是存活的命令

https://blog.51cto.com/waxyz/5336459

ping和telnet命令

telnet 命令常用于远程连接与管理目标主机，或查看某个目标主机的某个TCP端口是否开放。

ping 通常是用来检查源主机与目标主机网络是否通畅，或者测试网络连接质量。

springboot有什么优点

算法

反转链表