Spring中通过任务执行器TaskExecutor来实现多线程和并发编程。使用ThreadPoolTaskExecutor可实现一个基于线程池的TaskExecutor。因为实际开发中任务一般是异步的（即非阻塞的），所以要在配置类中@EnableAsync，并在实际执行的Bean方法中使用@Async来声明这是一个异步方法。

一、SpringBoot的线程池配置

创建配置类TaskExecutorConfig，用来定义ThreadPoolTaskExecutor，要使用@Configuration和@EnableAsync这两个注解，表示这是个配置类，并且使线程池的配置类。

@Slf4j

@Configuration  
@EnableAsync //开启异步任务支持  
public class TaskExecutorConfig {  
  
 @Value("${spring.async.corePoolSize}")  
 private int corePoolSize;  
 @Value("${spring.async.maxPoolSize}")  
 private int maxPoolSize;  
 @Value("${spring.async.keepAliveSeconds}")  
 private int keepAliveSeconds;  
 @Value("${spring.async.queueCapacity}")  
 private int queueCapacity;  
  
  
 @Bean("customizeExecutor")  
 public ThreadPoolTaskExecutor customizeExecutor() {  
 *log*.info("start asyncCustomizeExecutor");  
 ThreadPoolTaskExecutor taskExecutor = new ThreadPoolTaskExecutor();  
 // 设置线程名开头  
 taskExecutor.setThreadNamePrefix(DictInet.CUSTOMIZE\_TASK\_SIZE.*PREFIX*.getCode());  
 taskExecutor.setCorePoolSize(DictInet.CUSTOMIZE\_TASK\_SIZE.*CORE*.getCodeValue());  
 taskExecutor.setMaxPoolSize(DictInet.CUSTOMIZE\_TASK\_SIZE.*MAX*.getCodeValue());  
 taskExecutor.setKeepAliveSeconds(DictInet.CUSTOMIZE\_TASK\_SIZE.*SECONDS*.getCodeValue());  
 taskExecutor.setQueueCapacity(DictInet.CUSTOMIZE\_TASK\_SIZE.*QUEUE*.getCodeValue());  
 // 超过任务队列容量，被线程执行器拒绝提交的任务，由提交线程直接调用Runnable的run方法  
 ThreadPoolExecutor.CallerRunsPolicy callerRunsPolicy = new ThreadPoolExecutor.CallerRunsPolicy();  
 taskExecutor.setRejectedExecutionHandler(callerRunsPolicy);  
 return taskExecutor;  
 }  
  
}

注意，上面的方法名称为customizeExecutor，稍后马上用到。

二、将Service层的服务异步化

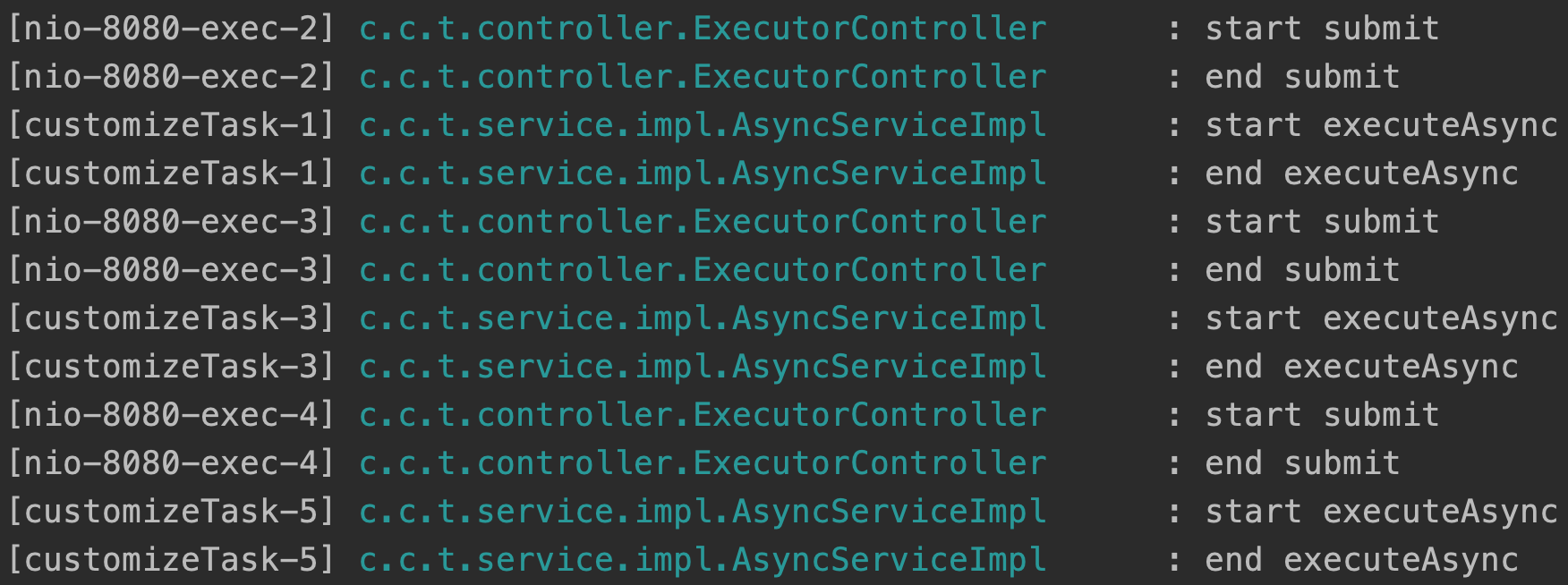
打开AsyncServiceImpl.java，在executeAsync方法上增加注解@Async(“customizeExecutor”)，customizeExecutor是前面配置文件中的方法名，表明executeAsync方法进入的线程池是customizeExecutor方法创建的，如下：

@Async

public void executeAsync() {  
 *log*.info("start executeAsync");  
  
 customizeExecutor.execute(() -> {  
 try {  
 Thread.*sleep*(1000);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
  
 *log*.info("end executeAsync");  
}

三、执行结果

如下日志中显示：



如上日志所示，controller的执行线程“nio-8080-exec-8”，这是tomcat的执行线程，而service层的日志显示线程名为“async-service-1”，显然已经在我们配置的线程池中执行了，并且每次请求中，controller的起始和结束日志都是连续打印的，表明每次请求都快速响应了，而耗时的操作都留给线程池中的线程去异步执行。

四、扩展ThreadPoolTaskExecutor

虽然用上了线程池，但是还不清楚线程池当时的情况，有多少线程在执行，多少在队列中等待？这里创建了一个ThreadPoolTaskExecutor的子类，在每次提交线程的时候都会将当前线程池的运行状况打印出来，代码如下：