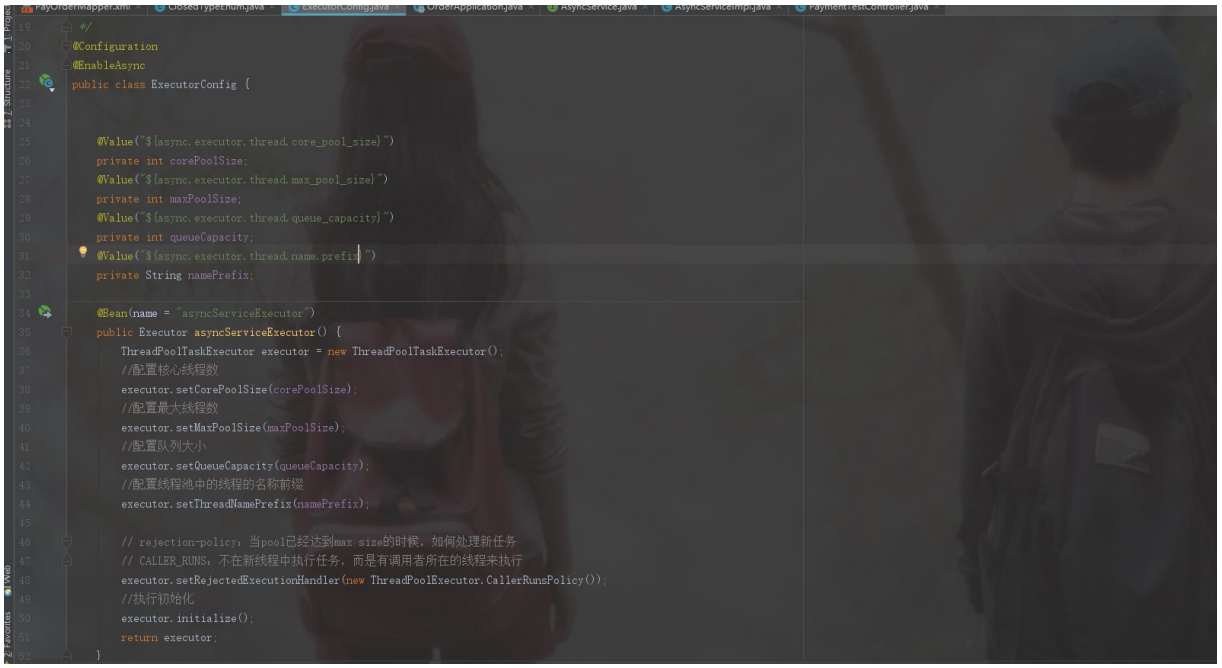
springboot 使用线程池创建异步方法调用

1. springboot时启动时添加ExecutorConfig配置类文件

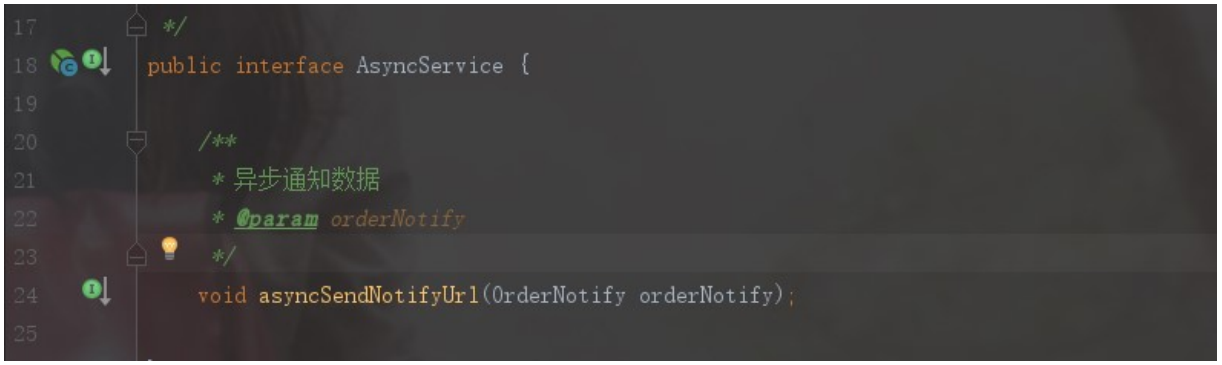


```
*/
@Configuration
@EnableAsync
public class ExecutorConfig {


    @Value("${async.executor.thread.core_pool_size}")
    private int corePoolSize;
    @Value("${async.executor.thread.max_pool_size}")
    private int maxPoolSize;
    @Value("${async.executor.thread.queue_capacity}")
    private int queueCapacity;
    @Value("${async.executor.thread.name.prefix}")
    private String namePrefix;

    @Bean(name = "asyncServiceExecutor")
    public Executor asyncServiceExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        //配置核心线程数
        executor.setCorePoolSize(corePoolSize);
        //配置最大线程数
        executor.setMaxPoolSize(maxPoolSize);
        //配置队列大小
        executor.setQueueCapacity(queueCapacity);
        //配置线程池中的线程的名称前缀
        executor.setThreadNamePrefix(namePrefix);

        // rejection-policy：当pool已经达到max size的时候，如何处理新任务
        // CALLER_RUNS：不在新线程中执行任务，而是有调用者所在的线程来执行
        executor.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());
        //执行初始化
        executor.initialize();
        return executor;
    }
```

2. 在启动类上面加入@EnableAsync 注解

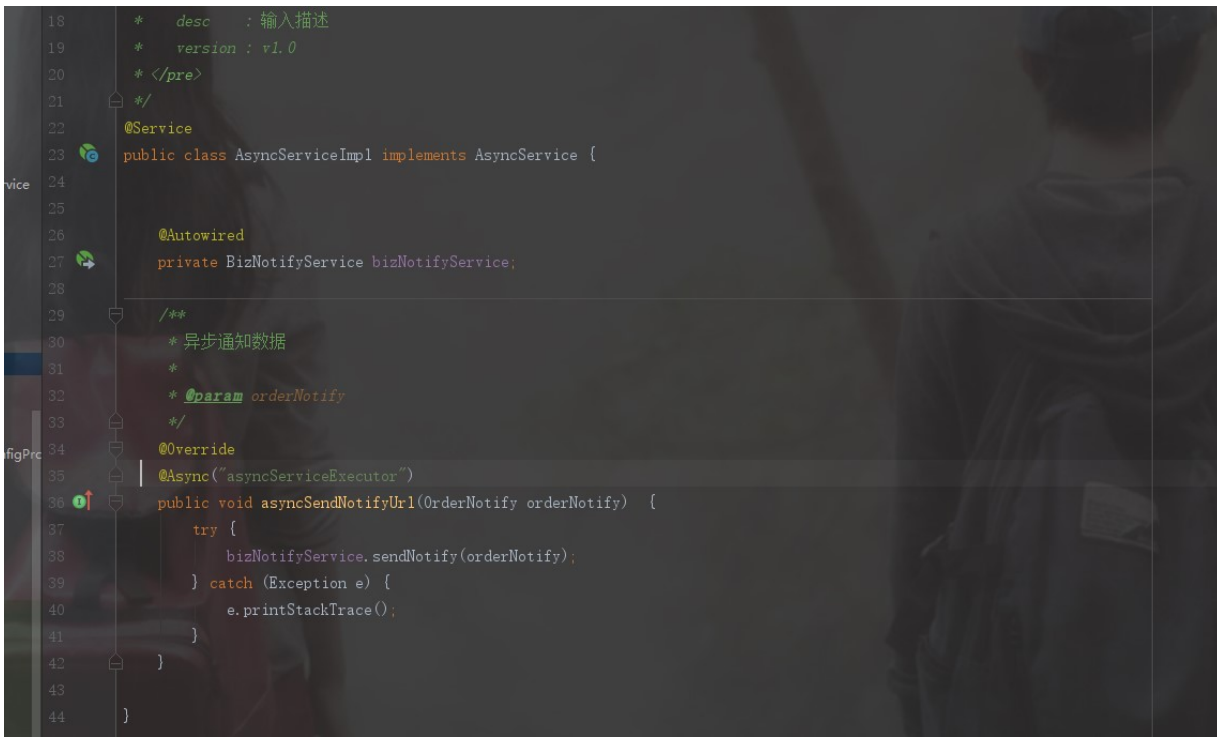
```
@EnableApolloConfig({"application","common.PayCommonConfig","localConfig"})
@EnableAsync  //线程池注解
public class OrderApplication  extends SpringMVCConfigureAdapter{
```

3. 创建异步线程接口和普通接口相同



```
*/
public interface AsyncService {

    /**
     * 异步通知数据
     * @param orderNotify
     */
    void asyncSendNotifyUrl(OrderNotify orderNotify);
```

4. 创建该接口的实现类，再需要异步处理的方法上添加@Async("xxxx") 注解即可



```
 *    desc    : 输入描述
 *    version : v1.0
 * </pre>
 */
@Service
public class AsyncServiceImpl implements AsyncService {


    @Autowired
    private BizNotifyService bizNotifyService;

    /**
     * 异步通知数据
     *
     * @param orderNotify
     */
    @Override
    @Async("asyncServiceExecutor")
    public void asyncSendNotifyUrl(OrderNotify orderNotify)  {
        try {
            bizNotifyService.sendNotify(orderNotify);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```