SpringBoot使用@Async注解8大坑点

juejin.cn/post/7279721061007654946

September 18, 2023

前言

> SpringBoot中，@Async注解可以实现异步线程调用，用法简单，体验舒适。

> 但是你一定碰到过异步调用不生效的情况，今天，我就列出90%的人都可能会遇到的8大坑点。

正文

1、未启用异步支持

> Spring Boot默认情况下不启用异步支持，确保在主配置类上添加`@EnableAsync`注解以启用异步功能。

java

复制代码

```
@SpringBootApplication
@EnableAsync
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

2、没有配置线程池

> 如果没有显式地配置线程池，Spring Boot将使用默认的SimpleAsyncTaskExecutor实现。

> 在生产环境，可能导致性能问题。建议使用自定义的线程池配置，推荐`ThreadPoolTaskExecutor`。

java

复制代码

```
@Configuration
@EnableAsync
public class AsyncConfig implements AsyncConfigurer {

    @Override
    public Executor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(10); // 设置核心线程数
        executor.setMaxPoolSize(100); // 设置最大线程数
        executor.setQueueCapacity(10); // 设置队列容量
        executor.setThreadNamePrefix("Async-"); // 设置线程名前缀
        executor.initialize();
        return executor;
    }

    // 其他配置方法...

}
```

3、异步方法在同一个类调用

> 异步方法必须是通过代理机制来触发的，因此如果在同一个类中调用异步方法，它将无法通过代理机制工作。

> 可以尝试将异步方法移到另一个Bean中，然后通过依赖注入进行调用，这也是万金油用法。

java

复制代码

```
// 你的业务服务
@Service
public class MyService {

    @Autowired
    private AsyncService asyncService;

    @Async
    public void asyncMethod() {
        // 异步方法逻辑...
        asyncService.asyncMethod(); // 在另一个Bean中调用异步方法
    }
}

// 你声明的异步服务，这里面可以是你所有的异步方法，哪里调用直接注入即可。
@Service
public class AsyncService {

    @Async
    public void asyncMethod() {
        // 异步方法逻辑...
    }
}
```

4、事务失效问题

> `@Async`方法默认不会继承父方法的事务。如果需要事务支持，请确保异步方法和调用该方法的方法都被`@Transactional`注解标记。

java

复制代码

```
@Service
public class MyService {

    @Autowired
    private MyRepository myRepository;

    @Async
    @Transactional
    public void asyncMethod() {
        // 异步方法逻辑...
        myRepository.save(entity);
    }
}
```

5、异常处理

> 异步方法中抛出的异常不能直接捕获，因为调用者将无法获取到异常。建议使用`Future`或`CompletableFuture`来捕获异步方法的异常并进行处理。

java

复制代码

```
@Service
public class MyService {

    @Async
    public CompletableFuture<String> asyncMethod() {
        try {
            // 异步方法逻辑...
            return CompletableFuture.completedFuture("Success");
        } catch (Exception e) {
            // 处理异常...
            return CompletableFuture.failedFuture(e);
        }
    }
}

// 调用异步方法并处理异常
CompletableFuture<String> future = myService.asyncMethod();
future.exceptionally(ex -> {
    // 异常处理逻辑...
    return "Error";
});
```

6、异步方法无返回结果

> 异步方法默认情况下是没有返回值的，如果需要获取异步方法的执行结果，依然要使用`Future`或`CompletableFuture`，可以将其设置为返回类型。

```
java

复制代码

@Service
public class MyService {

    @Async
    public CompletableFuture<String> asyncMethod() {
        // 异步方法逻辑...
        return CompletableFuture.completedFuture("Result");
    }
}

// 调用异步方法并获取结果
CompletableFuture<String> future = myService.asyncMethod();
String result = future.get(); // 阻塞等待结果
```

> 当然，正常情况下我们不需要返回结果，而且我也不建议这么干，异步线程本身也最适合处理不需要返回值的一类任务。

7、循环调用问题

> 当在同一个类中调用异步方法时，注意避免出现无限递归的循环调用。这可能会导致应用程序卡死或内存溢出。

```
java

复制代码

@Service
public class MyService {

    @Autowired
    private MyService myService; // 自身依赖

    @Async
    public void asyncMethod() {
        // 异步方法逻辑...
        myService.asyncMethod(); // 会导致无限递归调用
    }
}
```

> 这个坑点一般人不会遇到，但如果某些业务场景是关于树形结构的遍历、图算法等等，还是有几率出现这种情况的，这个坑点列出来仅供学习和了解。

8、异步方法顺序问题

> 异步方法的执行是非阻塞的，它们可能以任意顺序完成。如果需要按照特定的顺序处理结果，可以使用`CompletableFuture`的`thenApply`方法或者使用`@Async`的`order`属性来指定顺序。

java

复制代码

```
@Service
public class MyService {

    @Async("threadPoolTaskExecutor")
    public CompletableFuture<String> asyncMethod1() {
        // 异步方法1逻辑...
        return CompletableFuture.completedFuture("Result1");
    }

    @Async("threadPoolTaskExecutor")
    public CompletableFuture<String> asyncMethod2() {
        // 异步方法2逻辑...
        return CompletableFuture.completedFuture("Result2");
    }
}

// 调用异步方法并处理结果顺序
CompletableFuture<String> future1 = myService.asyncMethod1();
CompletableFuture<String> future2 = future1.thenCompose(
                        result1 -> myService.asyncMethod2());

String finalResult = future2.get(); // 阻塞等待最终结果
```

总结

> 这里面，我个人认为绝大多数人会遇到的坑点集中在没有配置自定义线程池、异步方法在同一个类中调用、事务不起作用这几个问题上。

> 所以，万金油的写法还是专门定义一个AsyncService，将异步方法都写在里面，需要使用的时候，就在其他类将其注入即可。