根据目前的趋势，我发现初级到高级的开发者都使用 Spring Boot 作为他们构建软件的首选务器。事实上，它对开发人员友好，它“约定优于配置”的风格有助于开发人员只关注业务逻辑。如果他们不确定 Spring 是如何运行的，只需要查看 Spring Boot 教程就可以开始开始使用 Spring Boot，就是这么简单。

我喜欢 Spring Boot 的另一部分是开发人员不必了解 Spring 的内部细节，只需添加一些注解，编写业务代码，看！虽说如此，有时，你必须知道它如何运行的。我想说的是，你需要更好的了解你的工具，这样你就可以像专业人士那样使用它。

在本文中，我将尝试让你更好的了解如何在 Spring 中使用异步处理。

任何与业务逻辑没有直接关联的逻辑片段（横切关注点）或在调用者上下文中不需要响应来确定下一个流程的逻辑或任何业务计算都是异步化的理想候选者。另外，在集成到分布式系统时，异步技术正在被用于让他们解耦。

在 Spring 中，我们可以使用 @Async 注解使用异步。但是如果你在方法顶部随机使用 @Async 并认为你的方法将在一个单独的线程中异步调用，那你就错了。你需要知道 @Async 如何运行以及他的局限性。没有这些，你无法理解异步行为。

@Async 如何运行？

当你在方法上放置 Async 注解时，它会根据 proxyTargetClass 属性定义的 Async（JDK Proxy/CGlib）创建该对象的代理。然后，Spring 尝试查找与上下文关联的线程池以提交该方法的逻辑作为独立的执行路径。确切的说，它搜索唯一的 TaskExecutorbean 或 被命名为 TaskExecutor 的 bean。如果没有找到，则使用默认的 SimpleAsyncTaskExecutor。

现在，当它创建一个代理并提交任务到 TaskExecutor 线程池，它有一些必须知道的限制。否则，你会对你的 Async 无法运行或创建一个新的线程而摸不到头脑！让我们来看看。

Async 的限制

1.假设你编写了一个类并确定一个方法担当 Async，并在该方法顶部放上了 @Async。现在，如果你想通过创建本地实例从另一个类使用该类，那么它不会触发异步。它必须被 Spring 的 @ComponentScan 注解收录或在一个被 @Configuration 标记的类里创建。

**在类里使用 Async 注解**

package com.example.ask2shamik.springAsync.demo;

import java.util.Map;

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Component;

@Component

public class AsyncMailTrigger {

@Async

public void senMail(Map<String,String> properties) {

System.out.println("Trigger mail in a New Thread :: " + Thread.currentThread().getName());

properties.forEach((K,V)->System.out.println("Key::" + K + " Value ::" + V));

}

}

**调用类**

package com.example.ask2shamik.springAsync.demo;

import java.util.HashMap;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class AsyncCaller {

@Autowired

AsyncMailTrigger asyncMailTriggerObject;

public void rightWayToCall() {

System.out.println("Calling From rightWayToCall Thread " + Thread.currentThread().getName());

asyncMailTriggerObject.senMail(populateMap());

}

public void wrongWayToCall() {

System.out.println("Calling From wrongWayToCall Thread " + Thread.currentThread().getName());

AsyncMailTrigger asyncMailTriggerObject = new AsyncMailTrigger();

asyncMailTriggerObject.senMail(populateMap());

}

private Map<String,String> populateMap(){

Map<String,String> mailMap= new HashMap<String,String>();

mailMap.put("body", "A Ask2Shamik Article");

return mailMap;

}

}

在这里，我创建了两个方法——一个使用被 @ComponentScann 收录的 AsyncMailtrigger 的 @Autowired 版本，但是在 WrongWayToCall 方法，我在本地创建了对象，所以它不会被 @ComponentScan 收录，因此，它不会产生新的线程并将在主线程中执行。

**输出**

Calling From rightWayToCall Thread main

2019-03-09 14:08:28.893 INFO 8468 --- [ main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'

Trigger mail in a New Thread :: task-1

Key::body Value ::A Ask2Shamik Article

++++++++++++++++

Calling From wrongWayToCall Thread main

Trigger mail in a New Thread :: main

Key::body Value ::A Ask2Shamik Article

2.永远不要在私有方法上使用 @Async。在运行时，它将无法创建代理，因此无法工作。

@Async

private void senMail() {

System.out.println("A proxy on Private method " + Thread.currentThread().getName());

}

3.永远不要在同一个类中写 Async 方法，在 caller 方法执行相同 Async 方法的同一个类中 caller 方法调用相同的 Async methodAsync 方法。因此，请记住，在使用该引用时，Async 不起作用，因为在这种情况下，虽然它创建了代理，但是调用会绕过代理并直接调用该方法，所以不会生成 Thread。这将阻止开发人员错误的假设它将以 Async 方式运行。大多数开发者都不小心以这种方式实现 Async，因此在编写 Async caller方法时要非常小心。在调用 Async 方法时，它应该在不同的类中。

RU5ErkJggg==.gif

**示例**

package com.example.ask2shamik.springAsync.demo;

import java.util.HashMap;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Component;

@Component

public class AsyncCaller {

@Autowired

AsyncMailTrigger asyncMailTriggerObject;

public void rightWayToCall() {

System.out.println("Calling From rightWayToCall Thread " + Thread.currentThread().getName());

asyncMailTriggerObject.senMail(populateMap());

}

public void wrongWayToCall() {

System.out.println("Calling From wrongWayToCall Thread " + Thread.currentThread().getName());

this.senMail(populateMap());

}

private Map<String,String> populateMap(){

Map<String,String> mailMap= new HashMap<String,String>();

mailMap.put("body", "A Ask2Shamik Article");

return mailMap;

}

@Async

public void senMail(Map<String,String> properties) {

System.out.println("Trigger mail in a New Thread :: " + Thread.currentThread().getName());

properties.forEach((K,V)->System.out.println("Key::" + K + " Value ::" + V));

}

}

4.最后一条建议是执行应用程序。请注意，我们使用了 @EnableAsync 注解。有了这个，Spring 在后台线程池中提交 @Async 方法。该类可以通过定义一个新的 bean 来自定义使用的 Executor。之后我将在第 2 部分通过一个示例展示如何做到这一点。

package com.example.ask2shamik.springAsync;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.scheduling.annotation.EnableAsync;

import com.example.ask2shamik.springAsync.demo.AsyncCaller;

@SpringBootApplication

@EnableAsync

public class DemoApplication {

@Autowired

AsyncCaller caller;

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public CommandLineRunner commandLineRunner(ApplicationContext ctx) {

return args -> {

caller.rightWayToCall();

Thread.sleep(1000);

System.out.println("++++++++++++++++");

Thread.sleep(1000);

caller.wrongWayToCall();

};

}

}

总结

好了，我希望你现在能理解 Async 内部如何工作的，以及它的一些限制。在我的下一篇文章中，我将讨论在 Async 中异常处理器（exception handler）如何工作。敬请期待！