تسجيل الدخول إلى □□□□□□□□ □□□□□□□□□□□ □□□□□□□

مثل عادي □□□□ كائن في ApplicationContext إلى للوصول □□□□□□ من ApplicationContextAware واجهة الاستخدام كيفية إليك العادي. كائنك و □□□□□ يديرها التي الفاصلات بين الفاصلات لتجسيد طريقة إلى تحتاج ،□□□□□ مستهلك بذلك: القيام

العام النهج

.□□□□□□ من تلقائيًا ApplicationContext باستقبال □□□□□□ يديرها لفاصلة ApplicationContextAware واجهة تسمح بواسطة إنشاؤها تم فاصلة ليس □أي □□□□□□ يديرها لا التي عادية □□□□ لفاصلة متاحًا السياق لتجعل هذا استخدام يمكنك محول □□□□□□□.

خطوات للوصول إلى □□□□□□□□□□□□□□□□□□□

1. **إنشاء فئة مساعدة يديرها □□□□□□**: حدد فئة تفعيل ApplicationContextAware وتديرها □□□□□□ ،مثلًا□ مع ملحق @Component□. ستستقبل هذه الفئة ApplicationContext عندما يبدأ تطبيق □□□□□□.

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;

@Component
public class ApplicationContextProvider implements ApplicationContextAware {
    private static ApplicationContext context;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) {
        context = applicationContext;
    }

    public static ApplicationContext getApplicationContext() {
        return context;
    }
}
```

   - □ @Component يضمن أن □□□□□□ يدير هذه الفاصلة.
   - □ setApplicationContext يتم استدعاؤه من قبل □□□□□□ لإدخال ApplicationContext.
   - □ context متغير ثابت ومتغير □□□□□□ يسمح بالوصول من أي مكان.

2. **الوصول إلى السياق في كائن □□□□ العادي**: في كائن □□□□ العادي□مثل مستهلك □□□□□ الذي تم إنشاؤه ،يدويًا□ استرجع ApplicationContext باستخدام فئة المساعدة واستخدمه للحصول على الفاصلات التي يديرها □□□□□□.

```java
public class MyKafkaConsumer {
    public void processMessage() {
        ApplicationContext context = ApplicationContextProvider.getApplicationContext();
        SomeService service = context.getBean(SomeService.class);
        //
    }
}
```

- السياق يجعل مما التشغيل، بدء عند □□□□□□ قبل من تهيئته يتم ApplicationContextProvider لأن هذا يعمل ثابت. بشكل متاحًا

3. استدعاء يمكنك، □□□□□□ يديرها فاصلة قبل من العادي □□□□ كائن إنشاء تم إذا :**صريح بشكل السياق تمرير البديل:** □□□□□□. متغير أو بناء عبر العادي الكائن إلى وتمريره الفاصلة تلك إلى ApplicationContext

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Component;

@Component
public class KafkaConsumerCreator {
    @Autowired
    private ApplicationContext context;

    public MyKafkaConsumer createConsumer() {
        return new MyKafkaConsumer(context);
    }
}

public class MyKafkaConsumer {
    private final ApplicationContext context;

    public MyKafkaConsumer(ApplicationContext context) {
        this.context = context;
    }

    public void processMessage() {
        SomeService service = context.getBean(SomeService.class);
        //
    }
}
```

- الاختبار. وقابلية وتحسين صريحة التبعية يجعل مما الثابتة، المتغيرات يتجنب هذا

□□□□□ بمستهلك خاص حل

إذا كنت تعمل مع مستهلك □□□□□ واستخدام □□□□□ □□□□□□، فإن النهج الموصى به هو دمج المستهلك مباشرة في نظام □□□□□□ عادي. □□□□ كائن معالجته من بدلاً

باستخدام @KafkaListener حدد مستهلك □□□□□ كفاصلة يديرها □□□□□□ عبر طريقة @KafkaListener. هذا يسمح لك باستدعاء ApplicationContext أو الفاصلات الأخرى مباشرة.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
public class MyKafkaConsumer {
    @Autowired
    private ApplicationContext context;

    @Autowired
    private SomeService someService;

    @KafkaListener(topics = "my-topic")
    public void consume(String message) {
        //            someService
        SomeService service = context.getBean(SomeService.class); //
        someService.process(message);
    }
}
```

- □ □□□□□ □□□□□□ يدير دورة حياة المستهلك، بما في ذلك التثبيت والتدوير.
- □ الحالات. معظم في يدويًا السياق استخراج إلى الحاجة يبعد الاستدعاء

مستهلك □□□□□ يدوي إذا كنت مضطرًا إلى إنشاء KafkaConsumer يدويًا، □مثلًا باستخدام مكتبة عميل □□□□□، □مباشرة أعلاه. للتمرير الصريح النهج في موضح هو كما إليها، ApplicationContext وتمرير □□□□□□ يديرها فاصلة داخل ذلك فافعل

التوصيات

- □ **لمستهلكات □□□□□**: استخدم @KafkaListener من □□□□□□ □□□□□ في فاصلة يديرها □□□□□□ كلما أمكن. هذه يستفيد من التكامل بين □□□□□□ و □□□□□ ويبسط الوصول إلى ApplicationContext و الفاصلات الأخرى.

- **لكائنات □□□□ العادية:** إما استخدم الثابت الناتج عن ApplicationContextProvider أو تمرير ApplicationContext الحالة عن يعبد لأنه ممكنًا، يكون عندما للتمرير الصريح الناتج في يفضل .□□□□□□ يدويا فاصلة من صريح بشكل الاختبار. قابلية وتحسين الثابتة

بما العادي، □□□□ لكائن متاحًا السياق لجعل فعال بشكل ApplicationContextAware استخدام يمكنك الطرق، لهذه بالتتبع الخاص. إعدادك حسب ،□□□□□ مستهلك ذلك في