访问 Spring 的 ApplicationContext

要使用 Spring 的 `ApplicationContextAware` 接口在普通的 Java 对象中访问 `ApplicationContext`，例如 Kafka 消费者，你需要一种方法来桥接 Spring 管理的 bean 与你的普通对象。以下是如何做到的：

一般方法

`ApplicationContextAware` 接口允许 Spring 管理的 bean 自动从 Spring 接收 `ApplicationContext`。你可以使用这种方法将上下文可用于一个不由 Spring 管理的普通 Java 对象（即不是由 Spring 容器创建的 bean）。

访问 ApplicationContext 的步骤

1. **创建一个 Spring 管理的辅助类**：定义一个实现 `ApplicationContextAware` 并由 Spring 管理的类（例如，使用 `@Component` 注解）。这个类在 Spring 应用启动时将接收到 `ApplicationContext`。

```
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` 确保 Spring 管理这个 bean。
   - `setApplicationContext` 由 Spring 调用以注入 `ApplicationContext`。
   - 静态的 `context` 变量和 getter 允许从任何地方访问。

2. **在你的普通 Java 对象中访问上下文**：在你的普通 Java 对象中（例如，手动创建的 Kafka 消费者），使用辅助类检索 `ApplicationContext`，并使用它来获取 Spring 管理的 bean。

```
public class MyKafkaConsumer {
    public void processMessage() {
```

```
        ApplicationContext context = ApplicationContextProvider.getApplicationContext();
        SomeService service = context.getBean(SomeService.class);
        // 使用服务或其他 bean
    }
}
```

- 这有效是因为 `ApplicationContextProvider` 在启动时由 Spring 初始化，使上下文静态可用。

3. **替代方案：显式传递上下文**：如果你的普通 Java 对象由 Spring 管理的 bean 创建，你可以将 `ApplicationContext` 自动注入到该 bean 中，并通过构造函数或 setter 将其传递给普通对象。

```
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);
        // 使用服务
    }
}
```

- 这避免了静态变量，使依赖关系显式化并提高了可测试性。

Kafka 消费者特定解决方案

如果你使用的是 Kafka 消费者并且使用的是 **Spring Kafka**，推荐的方法是将消费者直接集成到 Spring 生态系统中，而不是将其视为普通的 Java 对象。

使用 `@KafkaListener` 将你的 Kafka 消费者定义为一个带有 `@KafkaListener` 方法的 Spring 管理的 bean。这允许你直接自动注入 `ApplicationContext` 或其他 bean。

```
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) {
        // 直接使用 context 或 someService
        SomeService service = context.getBean(SomeService.class); // 如果需要
        someService.process(message);
    }
}
```

- Spring Kafka 管理消费者的生命周期，包括线程和轮询。
- 自动注入消除了在大多数情况下手动获取上下文的需求。

手动 Kafka 消费者 如果你必须手动创建 `KafkaConsumer`（例如，直接使用 Kafka 客户端库），请在 Spring 管理的 bean 中执行此操作，并将 `ApplicationContext` 传递给它，如上述显式传递方法所示。

推荐

- **对于 Kafka 消费者**：在 Spring 管理的 bean 中尽可能使用 Spring Kafka 的 `@KafkaListener`。这利用了 Spring 与 Kafka 的集成，并简化了访问 `ApplicationContext` 和其他 bean。

- **对于一般的普通 Java 对象**：要么使用静态的 `ApplicationContextProvider` 方法，要么从 Spring 管理的 bean 显式传递 `ApplicationContext`。当可行时，显式传递方法更为可取，因为它避免了静态状态并提高了可测试性。

通过遵循这些方法，你可以有效地使用 `ApplicationContextAware` 使 Spring 上下文可用于你的普通 Java 对象，包括 Kafka 消费者，具体取决于你的特定设置。