獲取 Spring 的應用程式上下文

要使用 Spring 的 `ApplicationContextAware` 接口在普通的 Java 物件（例如 Kafka 消費者）中存取 `ApplicationContext`，你需要一種方法來橋接 Spring 管理的 bean 與你的普通物件。以下是如何做到的：

一般方法

`ApplicationContextAware` 接口允許 Spring 管理的 bean 自動從 Spring 接收 `ApplicationContext`。你可以使用這個來讓上下文對於一個不由 Spring 管理的普通 Java 物件（即不是由 Spring 容器創建的 bean）可用。

存取 ApplicationContext 的步驟

1. **創建一個 Spring 管理的輔助類別**：定義一個實現 `ApplicationContextAware` 並由 Spring 管理的類別（例如標註為 `@Component`）。這個類別在 Spring 應用程式啟動時會接收到 `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` 確保 Spring 管理這個 bean。
   - `setApplicationContext` 由 Spring 請求來注入 `ApplicationContext`。
   - 靜態的 `context` 變數和 getter 允許從任何地方存取。

2. **在你的普通 Java 物件中存取上下文**：在你的普通 Java 物件（例如手動創建的 Kafka 消費者）中，使用輔助類別來獲取 `ApplicationContext`，並使用它來獲取 Spring 管理的 bean。

```
public class MyKafkaConsumer {
    public void processMessage() {
        ApplicationContext context = ApplicationContextProvider.getApplicationContext();
        SomeService service = context.getBean(SomeService.class);
        // 使用服務或其他 bean 如需
    }
}
```

- 這樣做是因為 `ApplicationContextProvider` 在啟動時由 Spring 初始化，使上下文靜態可用。

3. **替代方案：明確傳遞上下文**：如果你的普通 Java 物件由 Spring 管理的 bean 創建，你可以將 `ApplicationContext` 自動注入到該 bean 中，並通過構造函數或設置器將其傳遞給普通物件。

```
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);
        // 使用服務
    }
}
```

- 這樣可以避免靜態變數，使依賴明確並提高可測試性。

Kafka 消費者特定解決方案

如果你在使用 Kafka 消費者並使用 **Spring Kafka**，建議將消費者直接整合到 Spring 生態系統中，而不是將其視為普通 Java 物件。

使用 `@KafkaListener` 將你的 Kafka 消費者定義為一個 Spring 管理的 bean，並使用 `@KafkaListener` 方法。這樣可以直接自動注入 `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 管理消費者的生命週期，包括線程和輪詢。
- 自動注入消除了在大多數情況下手動獲取上下文的需求。

手動 Kafka 消費者 如果你必須手動創建 `KafkaConsumer`（例如使用 Kafka 客戶端庫直接），請在 Spring 管理的 bean 中進行，並將 `ApplicationContext` 傳遞給它，如上述明確傳遞方法所示。

建議

- **對於 Kafka 消費者**：在可能的情況下，使用 Spring Kafka 的 `@KafkaListener` 在 Spring 管理的 bean 中。這樣可以利用 Spring 與 Kafka 的整合，並簡化存取 `ApplicationContext` 和其他 bean。

- **對於一般的普通 Java 物件**：使用靜態的 `ApplicationContextProvider` 方法或從 Spring 管理的 bean 明確傳遞 `ApplicationContext`。當可行時，明確傳遞方法更受歡迎，因為它避免了靜態狀態並提高了可測試性。

通過這些方法，你可以有效地使用 `ApplicationContextAware` 使 Spring 上下文對你的普通 Java 物件（包括 Kafka 消費者）可用，具體取決於你的設置。