Spring Configuration Order

how to run spring confiuration before all configuration

In a Spring Boot application, if you want a specific configuration or initialization logic to run before all other configurations, you can follow one of these approaches:

1. Use `@Order` on Configuration Classes

- Spring allows you to specify the order in which configuration classes are processed using the `@Order` annotation.
- Add `@Order` to your configuration class to ensure it runs first.

```java
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;

@Configuration
@Order(1) // Ensures this configuration runs before others
public class EarlyConfiguration {
    // Define beans or initialization logic here
}
```

Configuration classes without an explicit order are processed after those with `@Order`.

2. Use `@DependsOn` in Beans

- If specific beans need to be initialized before others, use the `@DependsOn` annotation to enforce dependency ordering.

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class EarlyConfiguration {

    @Bean(name = "earlyBean")
    public String earlyBean() {
        return "Initialized early";
    }
}

@Configuration
public class DependentConfiguration {

    @Bean
    @DependsOn("earlyBean")
    public String dependentBean() {
        return "Initialized after earlyBean";
    }
}
```

3. Use a `SmartLifecycle` Bean

- Implement the `SmartLifecycle` interface for a bean that should start early in the application lifecycle.

```java
import org.springframework.context.SmartLifecycle;
import org.springframework.stereotype.Component;

@Component
public class EarlyLifecycleBean implements SmartLifecycle {

    private boolean running = false;

    @Override
    public void start() {
        // Initialization logic
        System.out.println("Early initialization logic");
        running = true;
    }

    @Override
    public void stop() {
        running = false;
    }

    @Override
    public boolean isRunning() {
        return running;
    }

    @Override
    public int getPhase() {
        return Integer.MIN_VALUE; // Runs in the earliest phase
    }
}
```

4. Use Application Listeners

- You can define an `ApplicationListener` to run logic during specific application events, such as `ApplicationEnvironmentPreparedEvent` or `ApplicationStartedEvent`.

```java
import org.springframework.boot.context.event.ApplicationStartedEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.stereotype.Component;

@Component
public class EarlyApplicationListener implements ApplicationListener<ApplicationStartedEvent
> {
    @Override
    public void onApplicationEvent(ApplicationStartedEvent event) {
        // Logic to run before other configurations
        System.out.println("Early initialization via ApplicationListener");
    }
}
```

5. Programmatic Configuration

- If you have control over the `SpringApplication` instance, you can register initializers or listeners programmatically in the `main` method.

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication app = new SpringApplication(Application.class);
        app.addListeners(new EarlyApplicationListener());
        app.run(args);
    }
}
```

6. Use `@Primary` or Explicit Ordering in Beans

- If multiple beans of the same type exist, use `@Primary` to make one take precedence.
- Alternatively, use `@Order` on method-level bean definitions to determine initialization order.

These approaches can help ensure your desired configuration or logic runs early in the application lifecycle. The choice depends on your specific requirements and whether you want to control the order of configurations, beans, or application startup events.