



Global / config / database

@ Configuration

해당 클래스가 스프링의 설정파일을 나타냄.
스프링 컨테이너에게 설정정보를 제공. 스프링빈들을
생성하고 구성하는 역할 수행.



'@Configuration'은 스프링 프레임워크에서 사용되는 어노테이션 중 하나로, 해당 클래스가 스프링의 설정 파일임을 나타냅니다.

'@Configuration' 어노테이션이 적용된 클래스는 스프링 컨테이너에게 설정 정보를 제공하고, 스프링 빈들을 생성하고 구성하는 역할을 수행합니다.

'@Configuration' 어노테이션을 사용하여 설정 클래스를 정의하면 다음과 같은 기능을 활용할 수 있습니다:

1. 빈(Beans) 등록: '@Bean' 어노테이션을 사용하여 메서드를 통해 스프링 빈을 등록할 수 있습니다. 이를 통해 개발자가 직접 객체를 생성하고 구성할 수 있으며, 스프링 컨테이너가 이를 관리하고 필요한 곳에서 주입할 수 있게 됩니다.
2. 외부 설정 파일 사용: '@PropertySource' 어노테이션과 함께 사용하여 외부 프로퍼티 파일을 로드하고, 설정 값들을 사용할 수 있습니다. 이를 통해 환경 변수, 프로퍼티 파일 등을 사용하여 애플리케이션의 설정을 유연하게 변경할 수 있습니다.
3. 다른 설정 클래스 임포트: '@Import' 어노테이션을 사용하여 다른 설정 클래스를 임포트할 수 있습니다. 이를 통해 여러 개의 설정 클래스를 조합하여 애플리케이션의 설정을 모듈화하고 관리할 수 있습니다.
4. 프로파일(Profile) 지정: '@Profile' 어노테이션을 사용하여 특정 환경이나 프로파일에 따라 빈을 조건부로 등록할 수 있습니다. 이를 통해 개발, 테스트, 운영 등 다양한 환경에 맞는 설정을 구성할 수 있습니다.

'@Configuration' 어노테이션을 사용하여 설정 클래스를 정의하면, 스프링 컨테이너는 해당 클래스를 로드하여 설정 정보를 읽고, 필요한 빈들을 생성하고 구성합니다. 이를 통해 스프링의 IoC(Inversion of Control) 기능을 활용하여 애플리케이션의 객체 관리와 의존성 주입을 편리하게 처리할 수 있습니다.

@ EnableJpaRepositories

스프링에서 JPA 레포지토리를 활성화하는데 사용되는 어노테이션
이 어노테이션은 스프링부트에서 자동으로 구성되지만, 일반적인
스프링 프로젝트에서는 수동으로 활성화해야 할 수도 있습니다.



'@EnableJpaRepositories'는 스프링에서 JPA(Java Persistence API) 리포지토리를 활성화하는 데 사용되는 어노테이션입니다. 이 어노테이션은 스프링 부트에서 자동으로 구성되지만, 일반적인 스프링 프로젝트에서는 수동으로 활성화해야 할 수도 있습니다.

'@EnableJpaRepositories' 어노테이션을 사용하면 다음과 같은 작업이 수행됩니다:

1. JPA 리포지토리를 사용하기 위한 스프링 빈들을 자동으로 구성합니다.
2. JPA 엔티티 클래스와 JPA 리포지토리 인터페이스가 위치한 패키지를 지정합니다.
3. JPA 리포지토리를 구현하는 빈들을 자동으로 생성하고 등록합니다.

'@EnableJpaRepositories' 어노테이션은 보통 스프링의 설정 클래스에 적용됩니다. 설정 클래스에 '@EnableJpaRepositories'를 추가하고, 'basePackages' 속성을 사용하여 JPA 리포지토리 인터페이스가 위치한 패키지를 지정합니다. 스프링은 해당 패키지과 그 하위 패키지에 위치한 JPA 리포지토리들을 자동으로 구성하여 스프링 컨테이너에 등록합니다.

'@EnableJpaRepositories' 어노테이션을 사용하여 JPA 리포지토리를 활성화하면, JPA 기반의 데이터 액세스 작업을 수행하는 데 필요한 구성과 빈들이 자동으로 제공되어 편리하게 사용할 수 있습니다. 이를 통해 개발자는 JPA 리포지토리를 작성하고, 스프링이 해당 리포지토리를 자동으로 구성하여 사용할 수 있게 됩니다.

@EnableTransactionManagement

스프링에서 트랜잭션 관리를 활성화 하려면 사용되는 어노테이션.

이 어노테이션을 사용하면 스프링은 트랜잭션 관리를 위한 기능 제공 스프링의 트랜잭션 기능을 사용할 수 있게 됩니다.



'@EnableTransactionManagement'은 스프링에서 트랜잭션 관리를 활성화하는 데 사용되는 어노테이션입니다. 이 어노테이션을 사용하면 스프링은 트랜잭션 관리를 위한 기능을 제공하고, 스프링의 트랜잭션 기능을 사용할 수 있게 됩니다.

'@EnableTransactionManagement' 어노테이션을 사용하면 다음과 같은 작업이 수행됩니다:

1. 트랜잭션 관리자 설정: 스프링 컨테이너에 트랜잭션 관리자 빈을 등록하고, 트랜잭션 관리자를 설정합니다. 일반적으로는 데이터베이스에 대한 트랜잭션을 관리하는 트랜잭션 관리자를 사용합니다.
2. 트랜잭션 어드바이스 적용: '@Transactional' 어노테이션이 적용된 메서드에 대해 트랜잭션 어드바이스를 적용합니다. 이를 통해 메서드 실행 전후에 트랜잭션을 시작하고 커밋 또는 롤백합니다.

'@EnableTransactionManagement' 어노테이션은 일반적으로 스프링의 설정 클래스에 적용됩니다. 설정 클래스에

'@EnableTransactionManagement'를 추가하면, 스프링 컨테이너는 트랜잭션 관련 기능을 활성화하고, '@Transactional' 어노테이션이 적용된 메서드에 대해 트랜잭션을 관리합니다.

'@EnableTransactionManagement' 어노테이션을 사용하여 트랜잭션 관리를 활성화하면, 스프링의 트랜잭션 기능을 사용하여 데이터베이스 트랜잭션을 관리할 수 있습니다. 이를 통해 데이터 일관성과 무결성을 유지하고, 예외 발생 시 롤백하여 데이터의 일관성을 보장할 수 있습니다.

@Configuration Properties

*.properties, *.yml 파일에 있는 property를 객체 클래스에 값을 가져와서 (바인딩) 사용할 수 있게 해주는 어노테이션.

Spring boot에서는 문법에 필요한 설정 (DB설정, LOG 설정 등) 들은

*.properties, *.yml에 써둔다.

@ConfigurationProperties는 스프링 프레임워크에서 외부 구성 속성을 Java 객체에 바인딩하기 위해 사용하는 주석입니다. 이 주석은 클래스나 클래스 내의 특정 메서드에 적용되며, 구성 파일(application.properties 또는 application.yml과 같은)에 정의된 속성을 Java 클래스의 필드 또는 메서드에 매핑할 수 있게 해줍니다.

@ConfigurationProperties를 사용하기 위해 다음 단계를 따라야 합니다:

1. 클래스 또는 메서드에 '@ConfigurationProperties' 주석을 추가합니다.
2. 바인딩할 속성들을 해당 클래스의 필드로 선언합니다. 필드 이름은 구성 파일에서의 속성 이름과 일치해야 합니다.
3. 구성 파일에 정의된 속성 값을 바인딩할 때 사용할 접두사(prefix)를 '@ConfigurationProperties' 주석의 value 속성으로 지정합니다. 예를 들어, '@ConfigurationProperties(prefix = "myapp")'라고 지정하면 "myapp" 접두사를 가진 속성들이 바인딩됩니다.
4. 구성 파일에 정의된 속성 값을 바인딩하기 위해 '@EnableConfigurationProperties' 주석을 사용하여 해당 클래스를 활성화시킵니다.
5. 구성 파일에 정의된 속성들을 읽어와 Java 객체에 바인딩하기 위해 '@ConfigurationProperties'가 적용된 클래스의 인스턴스를 생성합니다. 이렇게 생성된 인스턴스는 주입 또는 사용할 수 있습니다.

예를 들어, 다음은 'myapp' 접두사를 가진 속성들을 바인딩하는 예입니다:

```
java Copy code

@Configuration
@ConfigurationProperties(prefix = "myapp")
@EnableConfigurationProperties
public class MyAppProperties {
    private String property1;
    private int property2;
    // ...

    // Getters and Setters
}
```

위의 예에서 'MyAppProperties' 클래스는 'myapp.property1'과 'myapp.property2' 속성을 바인딩합니다. 이후에 이 클래스의 인스턴스를 주입하여 속성 값을 사용할 수 있습니다.

@ConfigurationProperties를 사용하면 외부 구성 파일에서 정의된 속성들을 쉽게 Java 객체에 매핑하여 사용할 수 있으므로, 스프링 애플리케이션의 구성을 관리하고 유지보수하기가 편리해집니다.

@Qualifier

사용할 의존 객체를 선택할 수 있도록 해준다.

- 1. 설정에서 bean의 환경과 값을 설정한다.
- 2. @Autowired 애노테이션이 적용된 주입 대상에 @Qualifier 애노테이션을 설정한다.
이때 @Qualifier의 값으로 앞서 설정한 환경값을 사용한다.

@Qualifier는 Spring Framework에서 동일한 인터페이스 또는 클래스의 여러 구현체 중에서 어떤 빈을 주입해야 하는지 명시적으로 지정하기 위해 사용하는 주석입니다. @Autowired 주석과 함께 사용되며, 어떤 빈이 주입되어야 하는지를 지정할 때 사용됩니다.

Spring 컨텍스트에 동일한 타입의 여러 빈이 존재할 때, @Qualifier 주석을 사용하여 어떤 빈이 주입되어야 하는지 구분할 수 있습니다. @Qualifier 주석은 빈에 대해 고유한 식별자를 제공하며, 주입 시 해당 식별자를 사용하여 어떤 빈을 선택해야 하는지 Spring에 알려줍니다.

@Qualifier 주석을 사용하는 방법은 다음과 같습니다:

- 1. 각 빈에 대해 고유한 식별자를 지정하는 @Qualifier 주석을 작성합니다. 이 식별자는 임의의 문자열로 지정할 수 있습니다.

```
java Copy code

@Component
@Qualifier("beanQualifier")
public class MyBean implements MyInterface {
    // Implementation
}
```

- 1. 주입 대상 필드나 메서드 파라미터에 @Qualifier 주석을 추가하여 어떤 빈을 주입받는지 명시합니다.

```
java Copy code

@Autowired
@Qualifier("beanQualifier")
private MyInterface myBean;
```

위의 예에서 @Qualifier("beanQualifier")는 "beanQualifier" 식별자를 가진 빈을 주입받도록 명시하고 있습니다.

@Qualifier 주석을 사용하면 Spring이 여러 구현체 중에서 어떤 빈을 선택해야 하는지 알 수 있습니다. 이를 통해 의존성 주입 시에 특정 빈을 명시적으로 선택할 수 있으며, 빈의 식별자를 통해 적절한 빈을 주입받을 수 있습니다.

