Auto-Configuration

권영태

Spring과 Spring Boot
뭐가 다른거지?

=> 의존성 버전 관리와 자동 설정

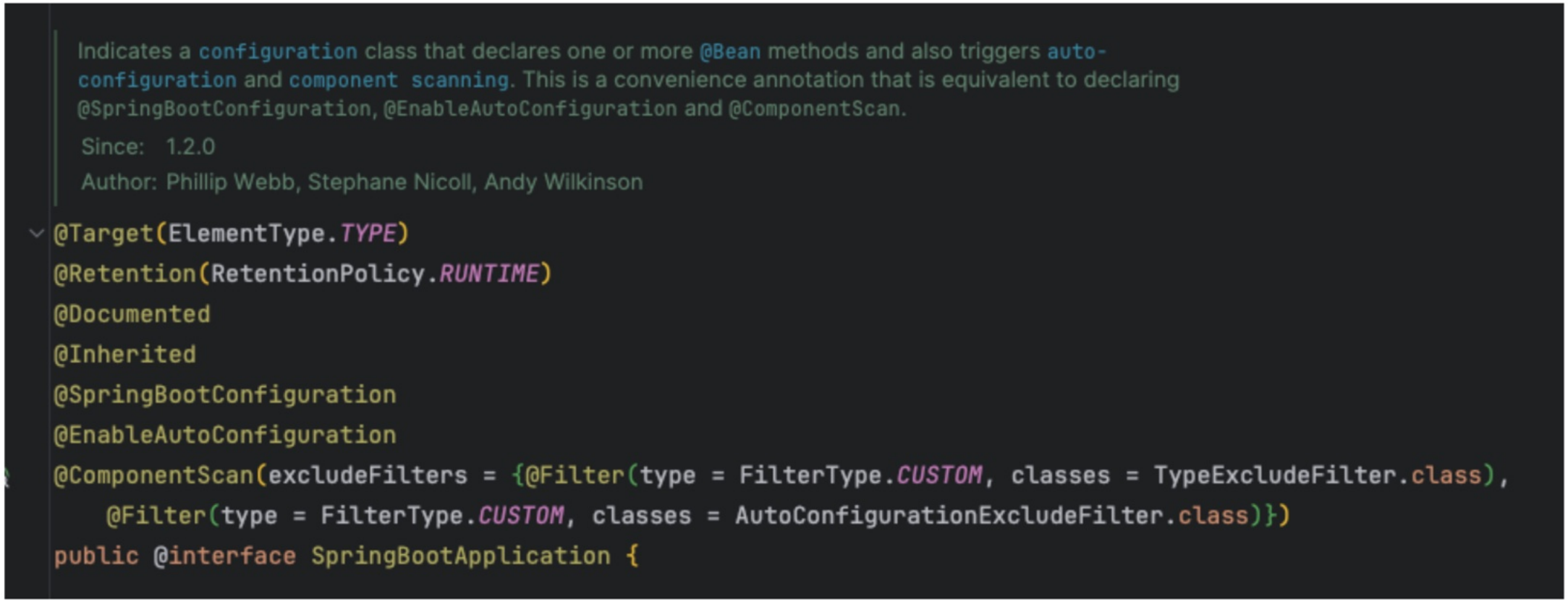
Spring Boot를 쓰는 우리는
DB Connection부터 Connection Poll 등
여러 bean 설정을 등록하지 않았다.
하지만 서버를 실행시키면 자연스럽게 사용할 수 있다.

도대체 어떻게 이런 일이 가능할까?

```java
@SpringBootApplication
public class MainApplication {
    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }
}
```

Spring Boot는 @SpringBootApplication 어노테이션이 붙은
Class에서만 서버를 실행시킬 수 있다.

이는 @SpringBootApplication 어노테이션 기반으로 동작한다는 말이다.



```
Indicates a configuration class that declares one or more @Bean methods and also triggers auto-
configuration and component scanning. This is a convenience annotation that is equivalent to declaring
@SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan.

Since: 1.2.0

Author: Phillip Webb, Stephane Nicoll, Andy Wilkinson

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class)})
public @interface SpringBootApplication {
```

@SpringBootConfiguraion, @EnableAutoConfiguration, @ComponentScan

@ComponentScan

패키지 하위에 있는 모든 @Respoistory, @Service 등 @Component 계열의 어노테이션이 붙은 클래스를 모두 빈으로 등록한다.

```
@ComponentScan(excludeFilters = {@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class)})
```

@SpringBootApplication에 있는 @CoponentScan을 보면 포함하지 않는 필터 2가지가 보인다.

이에 대한 내용은 @EnableAutoConfiguration에 대해 알아본 뒤 다시 살펴보자.

@SpringBootConfiguration

```
Author: Phillip Webb, Andy Wilkinson
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Configuration
@Indexed
public @interface SpringBootConfiguration {
```

@Configuration을 확장한 어노테이션인 @SpirngBootConfiguration

일반적으로 main()이 있는 클래스에 사용해 Spring Boot 메인 설정 클래스를 표시해준다.
이를 통해 Spring Boot가 자동 설정, 테스트 등에서 메인 설정 클래스를 명확하게 식별 할 수 있음!!

@EnableAutoConfiguration

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

Spring Boot에서 애플리케이션이 필요로 하는 설정들을 자동으로 구성해주는 핵심 어노테이션!!

이 어노테이션 덕분에 Spring과 달리 XML 또는 Conifg 등의 방식으로 설정을 하지 않아도,
설정 되지 않은 값들이 자동으로 설정된다.

@EnableAutoConfiguration

```
Creating new transaction with name [null]: PROPAGATION_REQUIRED,I
Acquired Connection [HikariProxyConnection@1225196709 wrapping co
Switching JDBC Connection [HikariProxyConnection@1225196709 wrapp
```

기본 적으로 Hirkari CP에서 Connection을 획득 => @EnableAutoConfiguration에서 기본 설정(Hirkari CP)으로 등록해서 사용

덕분에 개발자가 설정이 아닌 비즈니스 로직에 더욱 집중 할 수 있음!

@AutoConfigurationPackage

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

@AutoConfigurationPackage는 선언된 클래스 패키지와 그 하위 패키지들을 자동으로 스캔해 빈으로 등록시키기 위한 어노테이션

이는 AutoConfiguration이 동작할 때 도움을 주는 역할

@AutoConfigurationPackage

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Import(AutoConfigurationPackages.Registrar.class)
public @interface AutoConfigurationPackage {
```

내부적으로 @Import(AutoConfigurationPackages.Registrar.class) 즉, import한 클래스를 스프링 컨택스트에 등록

이 Registrar는 현재 클래스의 패키지 정보를 Bean 정의에 등록해주는 역할

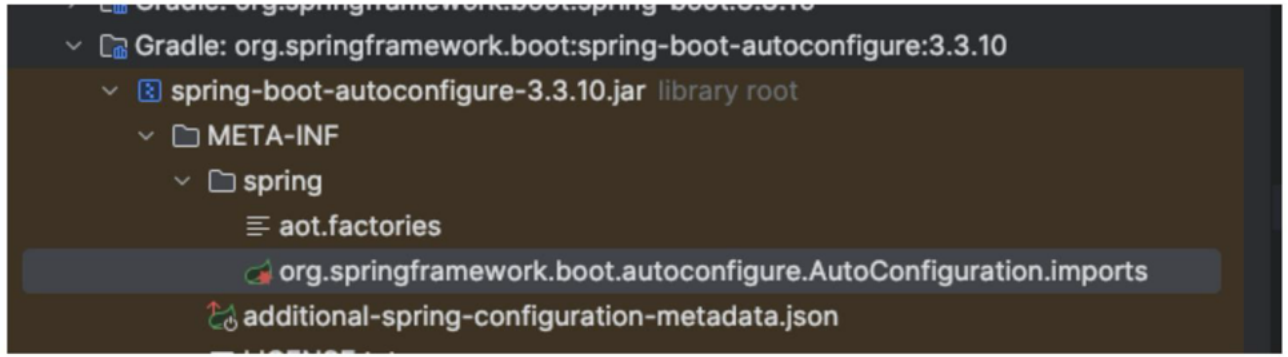
@AutoConfigurationImportSelector

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

자동 구성을 위한 핵심 역할을 하는 클래스(AutoConfigurationImportSelector)

AutoCofiguration.imports에서 자동 설정 후보 클래스 리스트를 읽고 이를 로드

AutoCofiguration.imports




```
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration
org.springframework.boot.autoconfigure.context.LifecycleAutoConfiguration
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration
org.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration
org.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration
org.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveDataAutoConfiguration
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveRepositoriesAutoConfiguration
```

@ComponentScan

```
@ComponentScan(excludeFilters = {@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class)})
```

다시 ComponentScan을 살펴보면 두 필터를 제외하는 것을 볼 수 있다.

그 중 AutoConfiguration이 관련된 두 번째 필터를 볼 수 있다.
이는 이미 자동 등록된 클래스가 중복으로 등록되지 않도록 하기 위함이다.

자동 설정이 아닌 직접 빈으로 등록 했다면?

1. 자동 구성을 진행하기 전 먼저 수행한 ComponentScan에 의해 스프링 컨테이너에 관련 빈이 등록되어 있는지 확인

2-1. 만약 등록되어 있다면, 자동 구성 후보 클래스를 스프링 컨테이너에 등록하지 않음

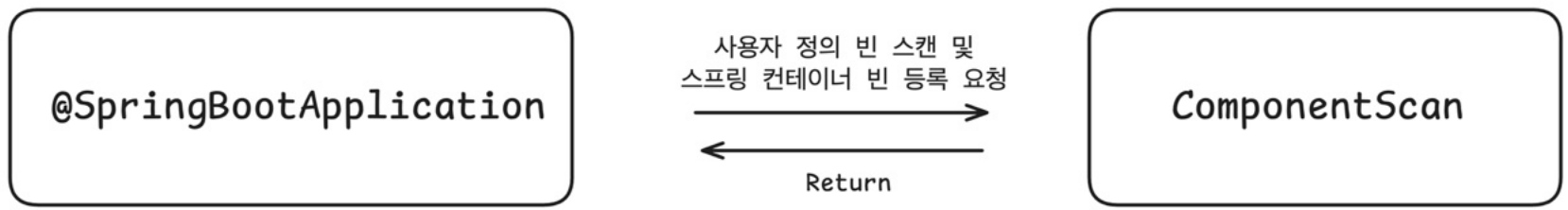
2-2. 만약 등되어 있지 않다면, 자동 구성 후보 클래스를 스프링 컨테이너에 등록

자동 설정이 아닌 직접 빈으로 등록 했다면?

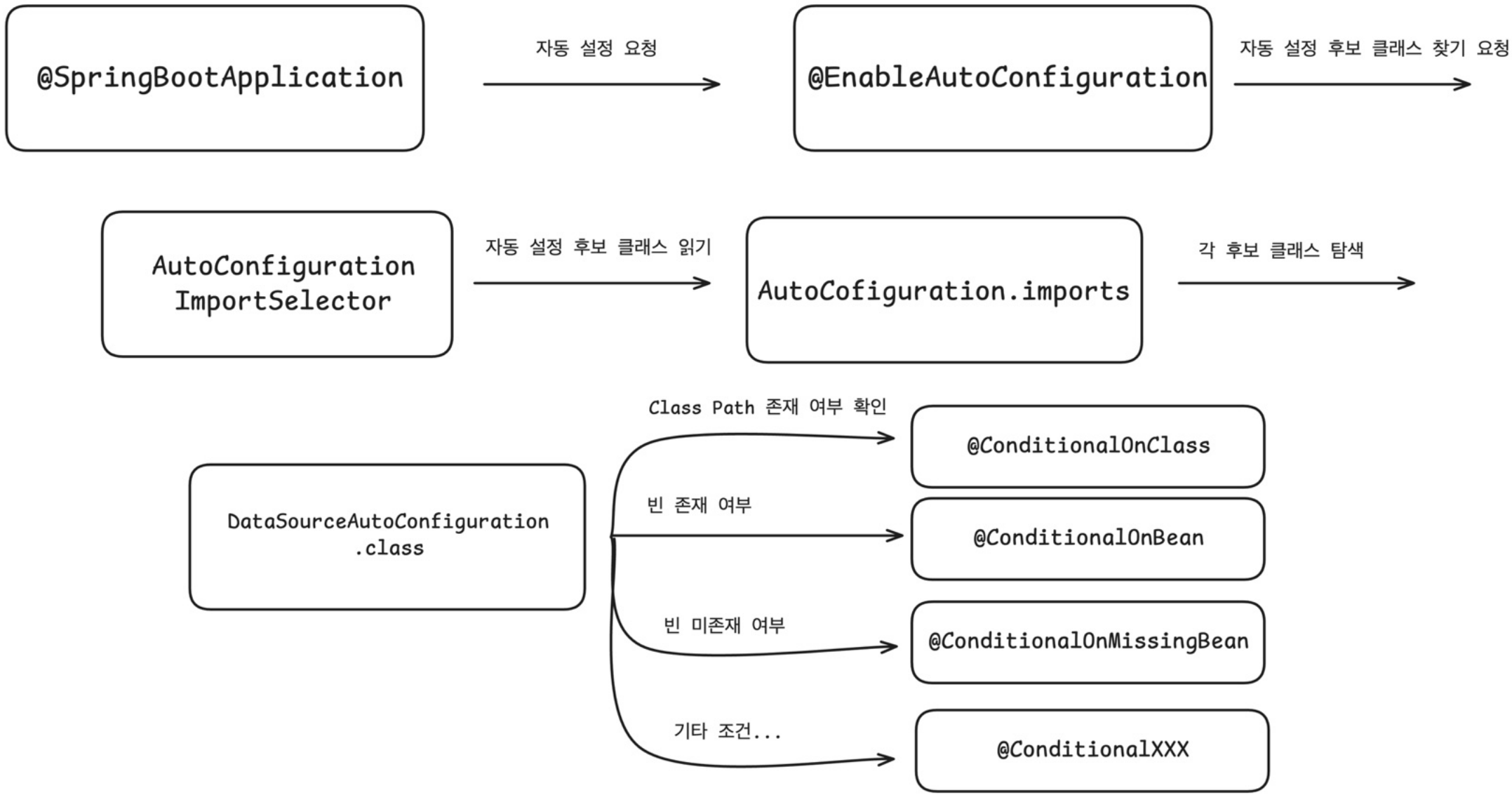
```java
@Slf4j
@SpringBootApplication
public class SpringtxApplication {

    new *
    public static void main(String[] args) {
        SpringApplication.run(SpringtxApplication.class, args);
    }

    new *
    @Bean
    public DataSource myDataSource() {
        log.info("myDataSource 등록!!!");
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setJdbcUrl("jdbc:h2:tcp://localhost/~/test");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

}
```

```
] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT
] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 18 ms. Fo
] hello.springtx.SpringtxApplication       : myDataSource 등록!!!
] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: defa
] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.5.3.Final
```

작동 순서(1)


@SpringBootApplication
사용자 정의 빈 스캔 및
스프링 컨테이너 빈 등록 요청
Return
ComponentScan

작동 순서(2)


@SpringBootApplication
자동 설정 요청
@EnableAutoConfiguration
자동 설정 후보 클래스 찾기 요청
AutoConfiguration
ImportSelector
자동 설정 후보 클래스 읽기
AutoCofiguration.imports
각 후보 클래스 탐색
DataSourceAutoConfiguration
.class
Class Path 존재 여부 확인
@ConditionalOnClass
빈 존재 여부
@ConditionalOnBean
빈 미존재 여부
@ConditionalOnMissingBean
기타 조건...
@ConditionalXXX