Chapter 04.스프링 부트 이해하기 - 03.스프링 부트 자동 설정

스프링 부트 자동 설정

1. 자동 설정이란?

- `자동 설정(Auto-configuration)`은 스프링 부트를 말할 때 빠질 수 없는 핵심 기술
- 자동 설정은 개발자들이 스프링 관련 프레임워크나 라이브러리를 추가했을 때 번거로운 설정 없이 동작하도록 최적화된 자동 설정을 내장하고 있는 것을 말한다
- 예를 들어 스프링 부트 프로젝트에 스프링 데이터 JPA, 스프링 데이터 몽고디비와 같은 프로젝트를 적용하면 내장된 설정이 동작하므로 개발자는 최소한의 설정 값(DB 접속 정보 등)만 넣어주거나 경우에 따라서는 설정을 새롭게 구성할 수 있다

---

2. 자동 설정 뜯어보기

- 이런 마법 같은 자동 설정을 위해 스프링 부트는 애플리케이션을 실행하는 시점에 `autoconfigure` 모듈 내부의 META-INF 하위에 있는 메타데이터 파일을 우선적으로 검색한다
- 메타데이터 파일은 이전에는 `META-INF/spring.factories` 였으나 현재는 `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` 이다
- org.springframework.boot.autoconfigure.AutoConfiguration.imports 내부

```
...
```

```
org.springframework.boot.autoconfigure.data.jdbc.JdbcRepositoriesAutoConfiguration
org.springframework.boot.autoconfigure.data.jpa.JpaRepositoriesAutoConfiguration
org.springframework.boot.autoconfigure.data.ldap.LdapRepositoriesAutoConfiguration
org.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfiguration
org.springframework.boot.autoconfigure.data.mongo.MongoReactiveDataAutoConfiguration
org.springframework.boot.autoconfigure.data.mongo.MongoReactiveRepositoriesAutoConfiguration
org.springframework.boot.autoconfigure.data.mongo.MongoRepositoriesAutoConfiguration

...
```

- 메타데이터 파일 내부에는 자동 설정을 구현한 `후보(Candidate)` 클래스들의 목록이 등록되어 있다
- 예를들어 (`jpa.HibernateJpaAutoConfiguration`, `redis.RedisRepositoriesAutoConfiguration`)
- 스프링 부트는 지난 시간에 공부한 `@EnableAutoConfiguration` 애노테이션이 존재하면 자동 설정 클래스를 검색하고 `조건에 따라서` 자동 설정 클래스를 로드한다
- 기본적으로 선언하지 않아도 @SpringBootApplication 애노테이션에 포함돼 있기 때문에 따로 선언할 필요가 없지만 직접 사용할 수 도 있다

```
@SpringBootConfiguration
@EnableAutoConfiguration(exclude=[JpaRepositoriesAutoConfiguration::class])
@ComponentScan(
  excludeFilters = [ComponentScan.Filter(
    type = FilterType.CUSTOM,
    classes = [TypeExcludeFilter::class]
  ), ComponentScan.Filter(type = FilterType.CUSTOM, classes = [AutoConfigurationExcludeFilter::class])]
)
// @SpringBootApplication // 주석처리
class HelloWorldApplication
```

---

3. 조건부 애노테이션

- 스프링 부트에는 많은 라이브러리에 대한 자동 설정 클래스가 준비되어 있다
- 자동 설정이 많다는 건 해야 할 설정이 적어지기 때문에 매우 편하지만 그렇다고 해서 사용하지도 않는데 모든 설정 클래스를 한 번에 로드하는 것은 불필요한 작업임
- 스프링 부트는 `조건부 애노테이션(Conditional Annotation)`인 `@Conditional` 사용해 조건에 따라 자동 설정 클래스를 로드한다
- spring-context 모듈 내부의 Conditional 애노테이션

```
package org.springframework.context.annotation;

public @interface Conditional {

  Class<? extends Condition>[] value();

}
```

- 또한 스프링 부트는 자주 사용되는 조건부 애노테이션을 재사용성을 위해 `미리 정의(Pre-Defined)` 하여 제공 하고 있습니다. 이러한 애노테이션은 `@ConditionalOn`이라는 이름으로 시작한다
- autoconfigure 모듈 내부의 ConditionalOn으로 시작하는 애노테이션

```
@Conditional(OnClassCondition.class)
public @interface ConditionalOnClass {}

@Conditional(OnPropertyCondition.class)
public @interface ConditionalOnProperty {}
```

- 클래스가 존재하는 경우에만 동작하거나 이와 반대로 클래스가 존재하지 않는 경우에만 동작한다
  - @ConditionalOnClass, @ConditionalOnMissingClass
- 빈이 애플리케이션 컨텍스트에 존재하는 경우에만 동작하거나 이와 반대로 빈이 애플리케이션 컨텍스트에 존재하지 않는 경우에만 동작한다
  - @ConditionalOnBean, @ConditionalOnMissingBean

- 다음은 미리 정의된 조건부 애노테이션이 어떤 식으로 사용되는지 알기 위해 스프링의 핵심 기능 중 하나인 AOP 자동 설정 클래스인 AopAutoConfiguration과 @ConditionalOnProperty를 예로 들어 설명한다
- 프로퍼티가 존재하는 경우에만 동작하는 `ConditionalOnProperty`가 적용된 자동 설정 클래스 `AopAutoConfiguration`

```
@AutoConfiguration
@ConditionalOnProperty(prefix = "spring.aop", name = "auto", havingValue = "true", matchIfMissing = true)
public class AopAutoConfiguration
```

- @ConditionalOnProperty의 첫번째 값인 `prefix`는 spring.aop로 시작하는 프로퍼티가 있는지를 검사하고 `name`은 prefix 뒤에 들어가는 프로퍼티의 이름
- `havingValue`는 프로퍼티의 값이 일치하면 동작

- 그러므로 AopAutoConfiguration 클래스는 프로퍼티가 아래와 같을 경우에 동작
- application.properties 코드 확인

```
spring.aop.auto=true
```

- 하지만 스프링은 기본적으로 aop 기반으로 동작하는데 그 이유는 해당 프로퍼티를 항상 선언하지 않더라도 필수적인 설정이라면 자동으로 설정을 로드 하는 방법으로 마지막에 선언된 `matchIfMissing = true` 사용
- matchIfMissing은 기본 값이 `false`이다 false인 경우에는 prefix, name, havingValue가 일치하는 경우에만 동작하지만 `true인 경우에는 해당 프로퍼티가 없더라도 동작하게 된다`