07 조건부 자동 구성

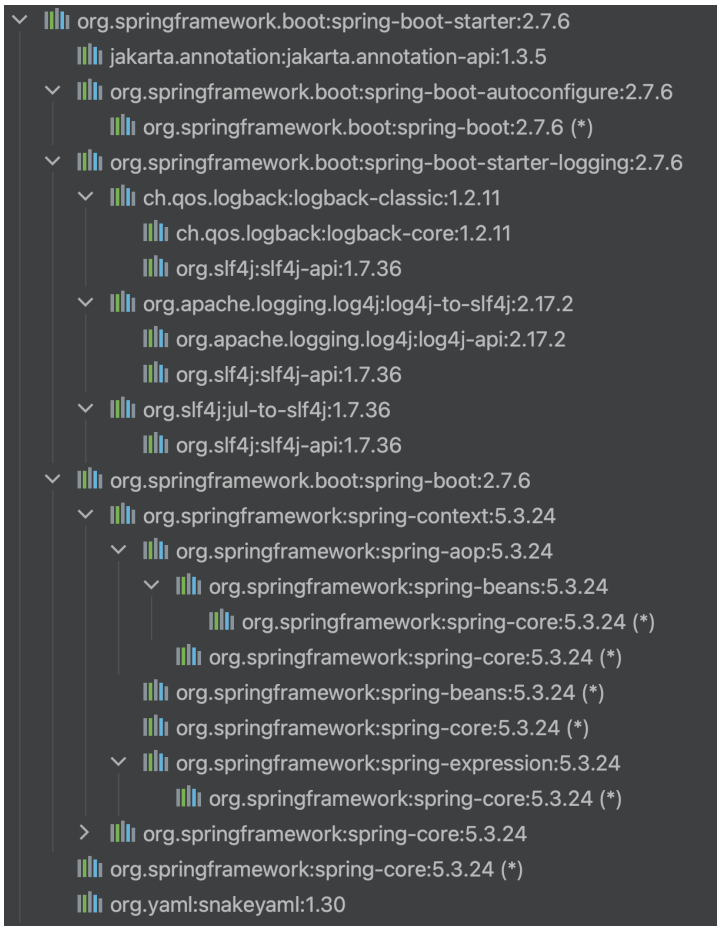
스타터와 Jetty 서버 구성 추가

스프링 부트의 Starter는 애플리케이션에 포함시킬 의존 라이브러리 정보를 담고 있다.

Maven 또는 Gradle의 의존 라이브러리 목록에 추가해서 스프링 부트가 선정한 기술의 종류와 버전에 해당하는 라이브러리 모듈을 프로젝트에 포함시킨다.

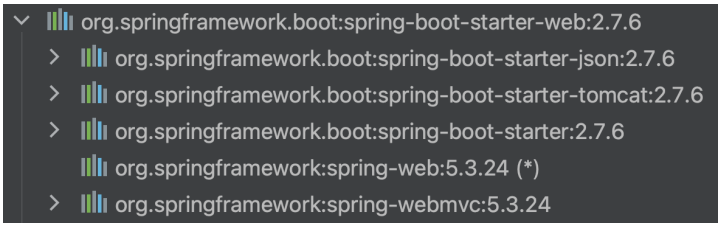
spring-boot-starter

가장 기본이 되는 스타터이다. 스프링 코어, 스프링 부트 코어를 포함해서 자동 구성, 애노테이션, 로깅 등에 필요한 의존 라이브러리가 포함되어있다.



spring-boot-start-web

Spring Initializr에서 web 모듈을 선택하면 이 스타터가 추가된다. spring-boot-starter를 포함한다. SpringWeb, SpringMVC와 Json, Tomcat 라이브러리가 추가된다.



spring-boot-starter-jetty

Jetty 서블릿 컨테이너를 이용하는데 필요한 라이브러리로 구성된다.

@Conditional과 Condition

@Conditional은 스프링 4.0에 추가된 애노테이션으로 모든 조건을 만족하는 경우에만 컨테이너에 빈으로 등록되도록 한다.

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Conditional {

  /**
   * All {@link Condition} classes that must {@linkplain Condition#matches match}
   * in order for the component to be registered.
   */
  Class<? extends Condition>[] value();

}
```

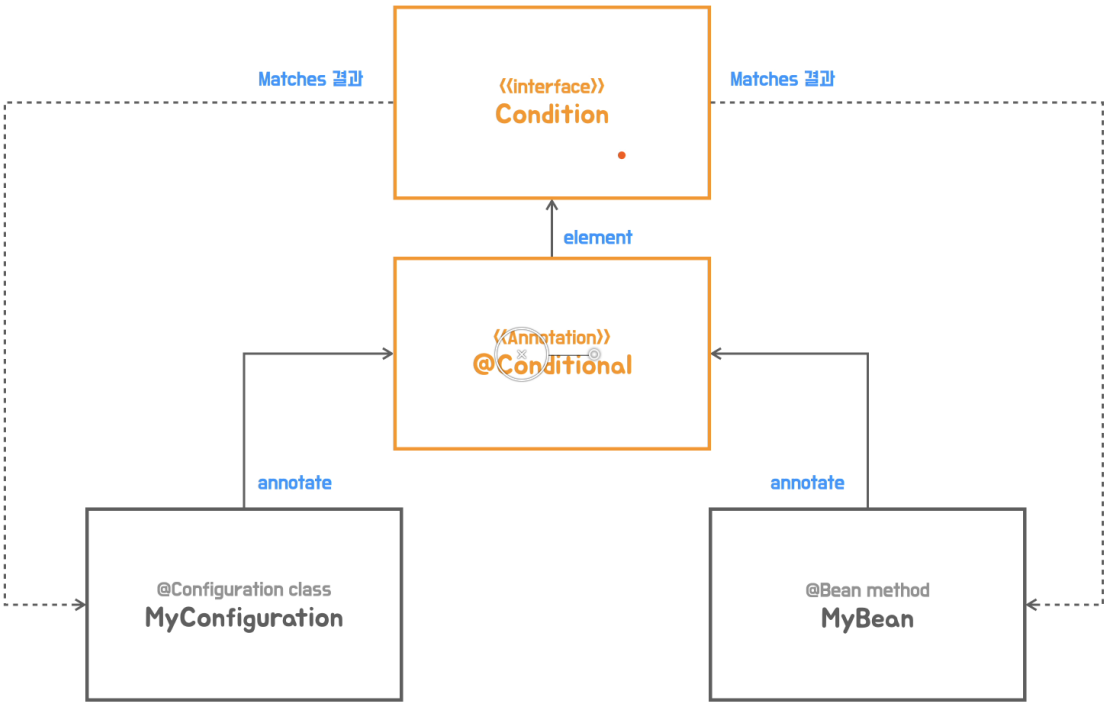
Condition은 @Conditional에 지정되어서 구체적인 매칭 조건을 가진 클래스가 구현해야할 인터페이스이다.

```
@FunctionalInterface
public interface Condition {

  /**
   * Determine if the condition matches.
   */
  boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);

}
```

@Conditional은 @Configuration 클래스와 @Bean 메소드에 적용 가능하다. 클래스 조건을 만족하지 못하는 경우 메소드는 무시된다.



스프링 부트가 제공하는 ApplicationContextRunner를 사용하면 스프링 컨테이너에 빈이 등록됐는지를 테스트 할 때 편리하다. @Conditional이 적용된 자동 구성 클래스의 적용 여부를 테스트 할 때 사용한다.

```
ApplicationContextRunner contextRunner = new ApplicationContextRunner();
    contextRunner.withUserConfiguration(Config1.class)
            .run(context -> {
                assertThat(context).hasSingleBean(MyBean.class);
                assertThat(context).hasSingleBean(Config1.class);
            });
```

```
new ApplicationContextRunner().withUserConfiguration(Config2.class)
        .run(context -> {
            assertThat(context).doesNotHaveBean(MyBean.class);
            assertThat(context).doesNotHaveBean(Config1.class);
        });
```

Condition의 matches 메소드에는 @Conditional 애노테이션의 엘리먼트 정보를 가져올 수 있는 AnnotatedTypeMetadata를 전달 받는다.

```
class BooleanCondition implements Condition {
    @Override
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        Map<String, Object> annotationAttributes = metadata.getAnnotationAttributes(BooleanConditional.class.getName());
        Boolean value = (Boolean)annotationAttributes.get("value");
        return value;
    }
}
```

커스톰 @Conditional

클래스 기준 조건부 구성

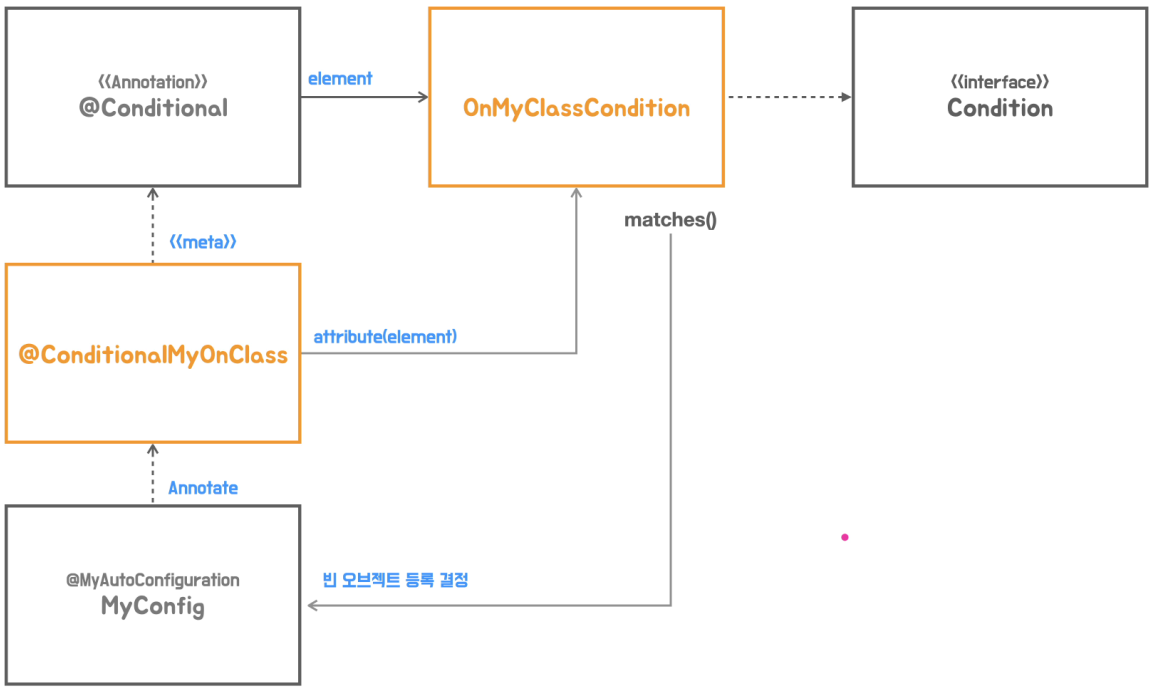
스프링 부트가 사용하는 @Conditional의 가장 대표적인 방법은 클래스의 존재를 확인하는 것이다. 스타터를 이용하거나 직접 의존 라이브러리 등록을 통해서 어떤 기술의 클래스를 애플리케이션이 사용하도록 포함시켰다면, 이 기술을 사용할 의도가 있다는 것으로 보고 관련 자동 구성 클래스를 등록시켜준다.

Tomcat과 Jetty 중에서 어떤 서블릿 컨테이너를 사용할지는 해당 서버 라이브러리 클래스가 프로젝트에 포함되어있는지를 확인하는 방법을 사용하면 된다.

특정 클래스가 현재 프로젝트에 포함되어서 클래스패스에 존재하는지 확인할 때는 스프링 ClassUtils.isPresent()를 사용하면 된다.

```
public class MyOnClassCondition implements Condition {
    @Override
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        Map<String, Object> attrs = metadata.getAnnotationAttributes(ConditionalMyOnClass.class.getName());
        String value = (String) attrs.get("value");
        return ClassUtils.isPresent(value, context.getClassLoader());
    }
}
```

커스톰 @Conditional을 사용할 때의 동작 방식은 아래와 같다.



자동 구성 정보 대체하기

자동 구성 정보는 다음의 과정으로 구성 정보가 등록된다

- imports 파일에서 자동 구성 정보 클래스 후보가 로딩된다
- @Conditional 조건 체크를 통해서 선택된 클래스가 빈으로 등록된다

@Conditional의 조건은 개발자가 프로젝트를 어떻게 구성하는지, 어떤 라이브러리가 포함되도록 하는지에 따라서 대부분 결정된다.

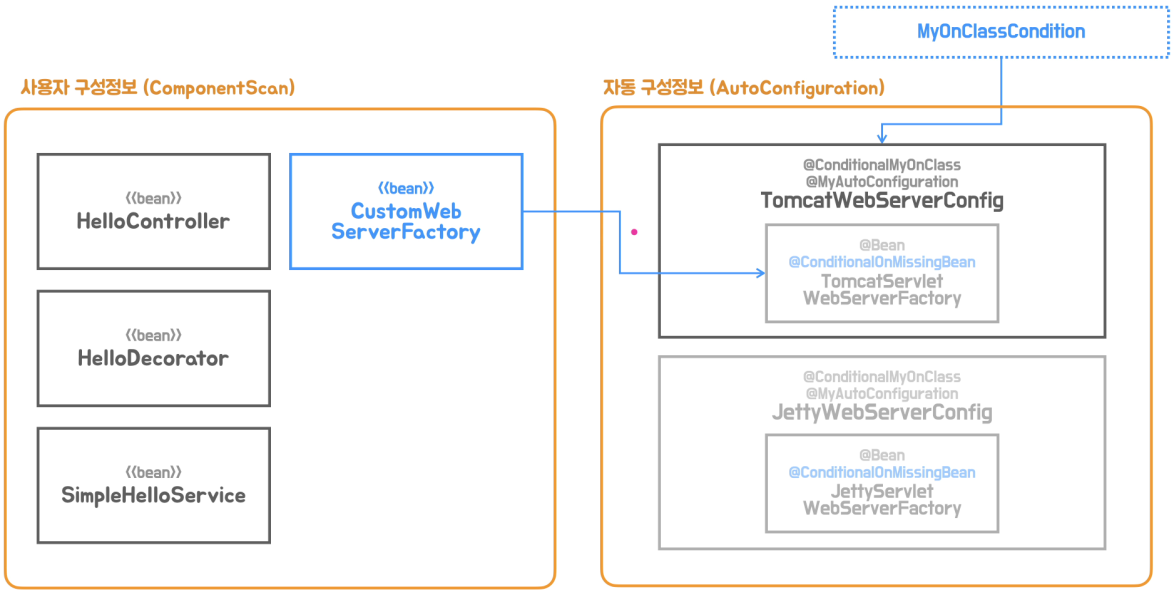
개발자가 자동 구성으로 등록되는 빈과 동일한 타입의 빈을 @Configuration/@Bean을 이용해서 직접 정의하는 경우 이 빈 구성이 자동 구성을 대체할 수 있다.

자동 구성 클래스의 @Bean 메소드에 @ConditionalOnMissingBean이 있는 경우엔 유저 구성에 지정한 타입의 빈이 정의되어있으면 자동 구성 빈의 조건이 충족되지 않아 등록되지 않는다.

```
@Bean("tomcatWebServerFactory")
@ConditionalOnMissingBean
public ServletWebServerFactory servletWebServerFactory() {
    return new TomcatServletWebServerFactory();
}
```

애플리케이션 코드에 다음과 같은 빈이 등록되어있으면 이게 우선이 된다.

```
@Configuration(proxyBeanMethods = false)
public class WebServerConfiguration {
    @Bean ServletWebServerFactory customerWebServerFactory() {
        TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();
        serverFactory.setPort(9090);
        return serverFactory;
    }
}
```



스프링 부트의 @Conditional

💡 스프링 프레임워크의 @Profile도 @Conditional 애노테이션이다.

`@Conditional(ProfileCondition.class)`

`public @interface Profile {`

스프링 부트는 다음과 같은 종류의 @Conditional 애노테이션과 Condition을 제공한다. 스프링 부트의 자동 구성은 이 @Conditional을 이용한다.

Class Conditions

- **@ConditionalOnClass**
- **@ConditionalOnMissingClass**

지정한 클래스의 프로젝트내 존재를 확인해서 포함 여부를 결정한다.

주로 @Configuration 클래스 레벨에서 사용하지만 @Bean 메소드에도 적용 가능하다. 단, 클래스 레벨의 검증 없이 @Bean 메소드에만 적용하면 불필요하게 @Configuration 클래스가 빈으로 등록되기 때문에, 클래스 레벨 사용을 우선해야 한다.

Bean Conditions

- **@ConditionalOnBean**
- **@ConditionalOnMissingBean**

빈의 존재 여부를 기준으로 포함여부를 결정한다. 빈의 타입 또는 이름을 지정할 수 있다. 지정된 빈 정보가 없으면 메소드의 리턴 타입을 기준으로 빈의 존재여부를 체크한다.

컨테이너에 등록된 빈 정보를 기준으로 체크하기 때문에 자동 구성 사이에 적용하려면 @Configuration 클래스의 적용 순서가 중요하다. 개발자가 직접 정의한 커스톰 빈 구성 정보가 자동 구성 정보 처리보다 우선하기 때문에 이 관계에 적용하는 것은 안전하다. 반대로 커스톰 빈 구성 정보에 적용하는 건 피해야 한다.

> 💡 @Configuration 클래스 레벨의 @ConditionalOnClass와 @Bean 메소드 레벨의 @ConditionalOnMissingBean 조합은 가장 대표적으로 사용되는 방식이다.
> 클래스의 존재로 해당 기술의 사용 여부를 확인하고, 직접 추가한 커스톰 빈 구성의 존재를 확인해서 자동 구성의 빈 오브젝트를 이용할지 최종 결정한다.

Property Conditions

@ConditionalOnProperty는 스프링의 환경 프로퍼티 정보를 이용한다. 지정된 프로퍼티가 존재하고 값이 false가 아니면 포함 대상이 된다. 특정 값을 가진 경우를 확인하거나 프로퍼티가 존재하지 않을 때 조건을 만족하게 할 수도 있다.

프로퍼티의 존재를 확인해서 빈 오브젝트를 추가하고, 해당 빈 오브젝트에서 프로퍼티 값을 이용해서 세밀하게 빈 구성을 할 수도 있다.

Resource Conditions

@ConditionalOnResource는 지정된 리소스(파일)의 존재를 확인하는 조건이다.

Web Application Conditions

- **@ConditionalOnWebApplication**
- **@ConditionalOnNotWebApplication**

웹 애플리케이션 여부를 확인한다. 모든 스프링 부트 프로젝트가 웹 기술을 사용해야 하는 것은 아니다.

SpEL Expression Conditions

@ConditionalOnExpression은 스프링 SpEL(스프링 표현식)의 처리 결과를 기준으로 판단한다. 매우 상세한 조건 설정이 가능하다.