10 스프링 부트 자세히 살펴보기

스프링 부트의 자동 구성과 테스트로 전환

직접 구현했던 자동 구성 클래스와 애노테이션 등을 제거한다.

메인 애노테이션을 스프링 부트의 @SpringBootApplication로 변경한다.

```
@SpringBootApplication
public class HellobootApplication {
```

테스트용으로 스프링 부트가 만들어주는 내장형 DB를 이용해서 데이터 액세스 로직만 테스트 할 때는 @JdbcTest를 사용할 수 있다. 프로퍼티로 설정한 DB 접속 정보가 사용되지 않으니 주의할 것.

스프링 컨테이너를 띄우고 자동 구성까지 적용해서 테스트를 할 때는 서블릿 컨테이너를 띄울 것인가에 따라서 다음 두 가지 방식을 사용할 수 있다.

롤백 테스트를 위해서는 @Transactional을 넣어줘야 한다.

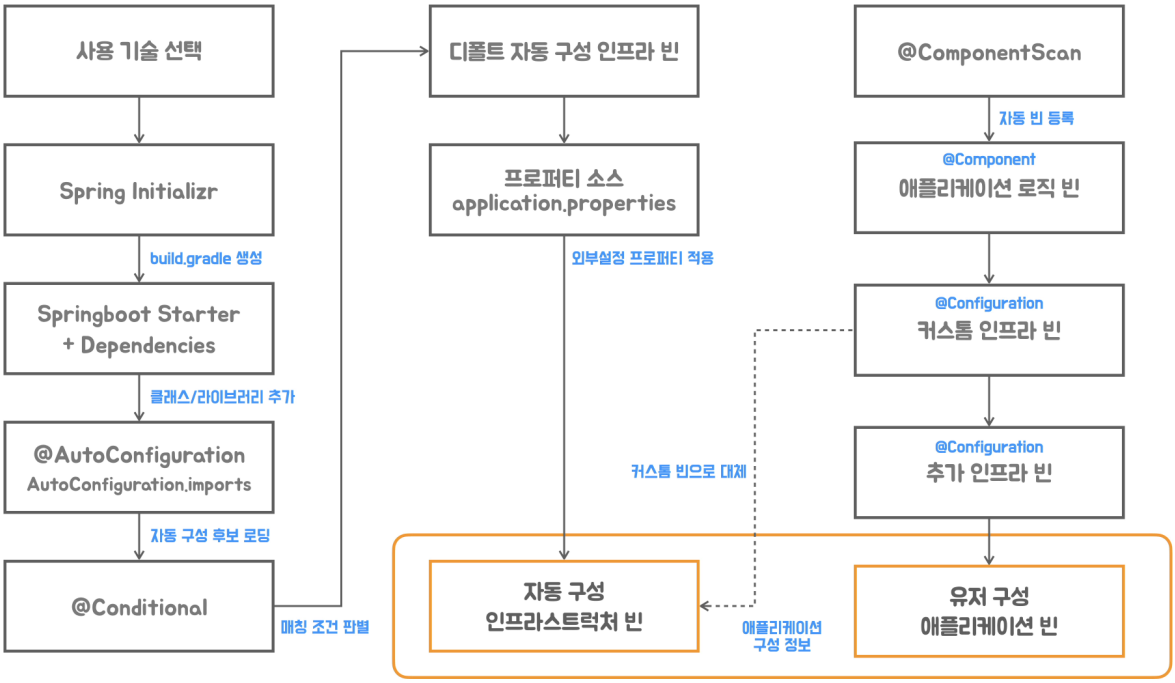
```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
@Transactional
public class HelloRepositoryTest {
```

서블릿 컨테이너를 띄워서 HTTP 요청을 보내서 테스트를 수행할 때는 다음과 같이 테스트를 준비한다.

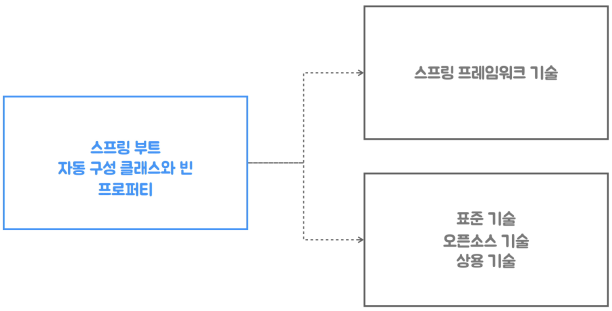
```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
public class HelloApiTest {
```

프로퍼티의 이름은 스프링 부트의 자동 구성에서 사용하는 것으로 변경해줘야 한다.

스프링 부트 자세히 살펴보기

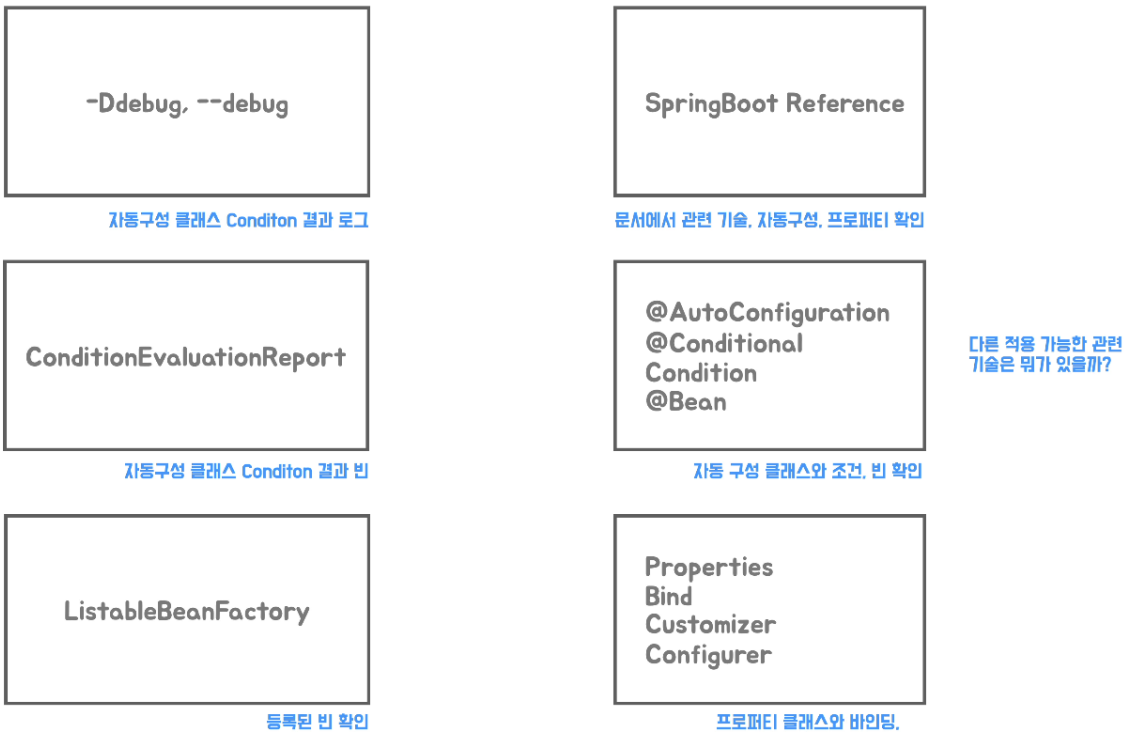


스프링 부트의 동작 방식을 이해하고, 자신이 사용하는 기술과 관련된 자동 구성과 프로퍼티 등을 분석하고, 어떻게 활용할 수 있는지 파악하는 것이 필요하다.



스프링 부트의 자동 구성으로부터 학습할 기술을 하나씩 찾아서 필요한 부분을 공부하는 접근 방법도 유용하다.

자동 구성 분석 방법



자동 구성 후보 목록과 조건 판단 결과를 조회하려면 -Ddebug이나 --debug 인자 값을 이용해서 스프링 부트 애플리케이션을 시작하면 된다.

코드를 이용하는 방법도 제공된다. ConditionEvaluationReport 타입의 빈을 주입 받아서 필요한 정보만 선택해서 자동 구성 결과를 보는 것도 가능하다.

최종적으로 등록된 빈 목록을 보고 싶으면 ListableBeanFactory 타입의 빈을 주입 받아서 빈 이름을 모두 가져오고, 필요하면 빈 오브젝트도 가져와서 살펴볼 수 있다.

자동 구성 선정 결과를 기준으로 스프링 부트 레퍼런스 문서, 그리고 자동 구성 클래스의 소스 코드, 프로퍼티 클래스, 커스토마이저 등을 차근차근 따라서 살펴보면서 어떻게 어떤 조건으로 동작할지 분석한다.

자동 구성 조건 결과 확인

ConditionEvaluationReport를 이용해서 조건이 매칭된 자동 구성 클래스와 메소드를 출력하는 코드를 작성할 수 있다. 그 중에서 Jmx 관련 구성 정보는 제외하고 싶으면 아래와 같이 코드를 작성하면 된다.

```
@Bean ApplicationRunner run(ConditionEvaluationReport report) {
      return args -> {
          System.out.println(report.getConditionAndOutcomesBySource().entrySet().stream()
                  .filter(co -> co.getValue().isFullMatch())
                  .filter(co -> co.getKey().indexOf("Jmx") < 0)
                  .map(co -> {
                      System.out.println(co.getKey());
                      co.getValue().forEach(c -> {
                          System.out.println("\t" + c.getOutcome());
                      });
                      System.out.println();
                      return co;
                  }).count());
        };
    }
```

Core 자동 구성 살펴보기

@ConditionalOnProperty 조건인 경우 matchIfMissing = true라면 프로퍼티가 존재하지 않아도 조건이 매칭된다.

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

Web 자동 구성 살펴보기

웹 스타터를 추가하기만 해도 50개 가까운 자동 구성 클래스와 빈 메소드 조건이 추가된다.

@ConditionalOnClass 등이 붙은 클래스에 @Import가 붙어있는 경우엔 클래스 레벨의 조건이 모두 만족하는 경우 @Import가 동작해서 자동 구성을 추가시기도 한다.

```
@AutoConfiguration(
    after = { GsonAutoConfiguration.class, JacksonAutoConfiguration.class, JsonbAutoConfiguration.class })
@ConditionalOnClass(HttpMessageConverter.class)
@Conditional(NotReactiveWebApplicationCondition.class)
@Import({ JacksonHttpMessageConvertersConfiguration.class, GsonHttpMessageConvertersConfiguration.class,
    JsonbHttpMessageConvertersConfiguration.class })
public class HttpMessageConvertersAutoConfiguration {
```

Customizer 빈을 이용하는 자동 구성은 프로퍼티 빈을 Customizer가 주입을 받고, 이를 빈 오브젝트를 만드는 메소드에서 Customizer를 주입 받아서 프로퍼티 설정 로직을 적용하는 방식으로 동작하기도 한다.

최종 빈 오브젝트를 만드는 Builder를 빈으로 등록하는 경우도 있다. 이 빌더 빈을 애플리케이션에서 가져다 사용해서 빈 오브젝트를 직접 구성하는 것도 가능하다.

RestTemplateBuilder 처럼 빌더만 자동 구성으로 제공하는 경우도 있다.

```
@Bean
@Lazy
@ConditionalOnMissingBean
public RestTemplateBuilder restTemplateBuilder(RestTemplateBuilderConfigurer restTemplateBuilderConfigurer) {
  RestTemplateBuilder builder = new RestTemplateBuilder();
  return restTemplateBuilderConfigurer.configure(builder);
}
```

Jdbc 자동 구성 살펴보기

DataSource 자동 구성에서 driver class name을 프로퍼티에 넣지 않으면 url을 이용해서 드라이버와 클래스 이름을 넣어준다. 내장형 DB를 사용하는 경우에 DataSource 프로퍼티가 없으면 스프링 부트가 자동으로 연결 정보를 설정해준다.

JdbcTransactionManager는 기존 DataSourceTransactionManager에 예외 추상화 작업의 변경이 반영되어있다. 기존 버전과 호환을 위해서 프로퍼티를 지정한 경우에만 선택된다.

@AutoConfiguration은 자동 구성을 진행할 전후 순서를 지정할 수 있다.

```
@AutoConfiguration(after = DataSourceAutoConfiguration.class)
```

TransactionProperties는 PlatformTransactionManagerCustomizer를 구현한다. 프로퍼티 클래스가 자신이 가진 값을 이용하는 간단한 Customizer 기능을 구현해서 사용하되기도 한다.

```
@ConfigurationProperties(prefix = "spring.transaction")
public class TransactionProperties implements PlatformTransactionManagerCustomizer<AbstractPlatformTransactionManager> {
```

정리

스프링 부트는

- 스프링 프레임워크를 잘 쓰게 도와주는 도구의 모음
- 서블릿 컨테이너와 관련된 모든 번거로운 작업을 감춰줌
- 스프링과 각종 기술의 주요 인프라스트럭처 빈을 자동 구성을 이용해서 자동으로 등록해줌
- 외부 설정, 커스톰 빈 등록을 통해서 유연하게 확장 가능

스프링 프레임워크

- 빈 오브젝트의 생명주기를 관리하는 컨테이너
- 빈 오브젝트의 의존 관계를 동적으로 주입해주는 어셈블러
- 구성 정보(configuration metadata)와 애플리케이션 기능을 담은 오브젝트가 결합되어 동작하는 애플리케이션이 된다
- @Configuration, @Bean, @Import를 이용한 구성 정보
- 메타 애노테이션, 합성 애노테이션 활용

스프링 부트 더 알아가기

- 스프링 부트의 코어 (Profile, Logging, Testing...)
- 핵심 기술 영역 (Web, Data, Messaging, IO...)
- 운영환경의 모니터링, 관리 방법
- 컨테이너, 배포, 빌드 툴
- 스프링 부트 3.x
- 스프링 프레임워크와 자바 표준, 오픈소스 기술