Spring

외부 properties 파일 실행방법

java -jar myproject.jar

--spring.config.location=classpath:/overrides.properties

생성자 주입방식을 사용해야하는 이유설명

아래링크 내용을 아주 간단하게 요약하면

1.컴파일에서 에러 걸러줄수있음

2.순환참조 방지가능

\*순환참조 : A라는 객체가 B에 의존하면서, B라는 객체가 A에 동시에 의존하고 있는경우.. 하나를 호출하면 서로 호출만하다가 stackoverflow떨어짐..

<https://yaboong.github.io/spring/2019/08/29/why-field-injection-is-bad/>

spring boot에서 외부 설정파일 읽어오는방법

-기본적으로 외부의 config폴더안의 하위디렉토리에다가 application.properties 이름으로 만들어놓으면 읽음..

<https://www.baeldung.com/spring-properties-file-outside-jar>

log4jdbc

- application에서 쿼리를 실행하면 로그를 깔끔하게 남겨준다.. 특정 패키지 안에서 실행되는 쿼리의 로그를 따로 분리하고 싶었으나, 쿼리에 관련된 로그를 통째로 관리해야하는듯함.. 즉 패캐지 별로 로그를 쪼개지는 못하는듯..

<https://www.leafcats.com/45>

<https://steven-life-1991.tistory.com/105>

Spring boot로 간단한 CRUD하는 프로젝트 설명(셋팅부터 배포까지..)

<https://daddyprogrammer.org/post/404/spring-boot2-design-api-interface-and-data-structure/>

Config 클래스에 bean으로 등록하면서 @ConfigurationProperties 라는 어노테이션을 함께 쓰기도한다. 이는 Spring boot2.x에서 나오기 시작하엿는데, application.properties파일을 통해 bean으로 등록해주기 위해 사용된다.

기존에 jar파일이나 스프링부트 내부에 등록되어있는(application.properties파일을 통해 바로 등록이 가능한.. ex)hikaricp ) 설정을 객체 생성해서 bean으로 올려줄수있도록해준다.

반면 application properteis에 적당한 prefix를 등록해놓으면 클래스 만들때, 클래스명 위에 어노테이션을 선언하면 설정에 맞추어 변수도 다 생성이 된다.

Hikaricp같은 경우는 기존에 springboot에 등록되어있는것이고, HikariConfig 클래스를 @ConfigurationProperties로 만들수 있는데, 이에 대한 prefix는 spring.datasource.hikari이다.(sping.datasource.hikari.XX 이런식으로 계속 등록을 해나가므로…)

아래는 참고사항(주석에 설명쓰여잇음)

@Configuration  
public class DbConfig {  
 @Autowired  
 private ApplicationContext applicationContext;  
  
  
 @Bean  
 @ConfigurationProperties(prefix = "spring.datasource.hikari")  
 public HikariConfig hikariConfig() {  
 return new HikariConfig();  
 }  
  
 /\*  
 \* //application.properties에 spring.datasource.hikari.jdbc-url 와 같은 db설정을 가능하게 해주는 속성셋팅값들이 제공..  
 \* 즉, spring.datasource.hikari 의 객체가 필요할텐데, 이 객체를 생성하는 부분을 위의 설정으로 가능..  
 \* 설정 객체 만들고, 아래 HikariDataSource의 생성자 파라미터로 config 넣어준다..  
 \*  
 \* ####참고사항 : Bean으로 등록되어지면 hikariConfig()함수를 호출할때 new HikariConfig()로 계속 객체를 생성할지라도 하나만생성! 싱글톤!  
 \* 기본적으로 싱글톤이다.. @Scope를 지정하면 계속 호출할때마다 새로 생성할수도있음..  
 \*  
 \* \*/  
  
  
  
 @Bean  
 public DataSource dataSource() {  
 DataSource dataSource = new HikariDataSource(hikariConfig());  
 return dataSource;  
 }  
  
  
 @Bean  
 public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {  
 SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();  
 sqlSessionFactoryBean.setDataSource(dataSource);  
 sqlSessionFactoryBean.setMapperLocations(applicationContext.getResources("classpath:/mapper/\*\*/\*.xml"));  
 return sqlSessionFactoryBean.getObject();  
 }  
  
 @Bean  
 public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) {  
 return new SqlSessionTemplate(sqlSessionFactory);  
 }  
}

참고사이트 : <https://velog.io/@max9106/Spring-Boot-%EC%99%B8%EB%B6%80%EC%84%A4%EC%A0%95-4xk69h8o50>

순환참조

생성자에 넣었을경우 @Lazy 어노테이션을 넣어준다..

혹은 setter로 넣어준다..

<https://hungrydiver.co.kr/bbs/detail/develop?id=90&scroll=comment>

=>혹여나 클래스간 데이터를 주고받을 필요가있다면, applicaionConext를 통해서 bean들을 모두 불러와서 listenIntf or fireIntf를 구현한 bean들만 메세지 송수신 가능하도록(Observer패턴) 셋팅해준다..

MDC 사용방법 잘 정리

1) <https://jeong-pro.tistory.com/199>

2) <https://m.blog.naver.com/PostView.nhn?blogId=sung487&logNo=221314283286&proxyReferer=https:%2F%2Fwww.google.com%2F>

인터셉터 관련

-token과 같은 인증관련 사용하기좋음

-controller로 넘어가기전에 처리해주는 prehandle 유용하게 쓸수있음

-controller에서 view 랜더링 전에 수행되는 posthandle도 있음

https://jhkang-tech.tistory.com/53

+추가로 예외관련해서 정리 잘해놓은 사이트

<https://eblo.tistory.com/48>

hikaricp 설정값관련 정리굿

<https://2ssue.github.io/programming/HikariCP-MySQL/>

[2021-03-02 16:37:56] [ WARN] [XNIO-1 task-2] PoolBase isConnectionAlive HikariPool-1 - Failed to validate connection org.mariadb.jdbc.MariaDbConnection@467508f1 (Connection.setNetworkTimeout cannot be called on a closed connection). Possibly consider using a shorter maxLifetime value.

관련 로그 내용 설명 굿 : https://jaehun2841.github.io/2020/01/08/2020-01-08-hikari-pool-validate-connection/#mysql%EC%9D%98-wait\_timeout