3.x

DataSource AutoConfiguration

=====:

==========

Spring boot 2.x uses the following algorithm while creating DataSource obj through AutoConfiguration process when relavent starters/jar files are added (spring-boot-starter-jdbc or

(1) Hikari cp DataSource

(2) Tomcat cp (if hikari cp jar file is not avalable)

spring-boot-starter-jpa

(3) Apache dbcp2 (if hikari cp, tomcat cp jar files are not avalable)

(4) Oracle UCP (if hikari cp, tomcat cp jar, apache dbcp jar files are not avalable)

To enable Tomcat CP DataSource Spring boot App

using

step1) Exclude Hikaricp jar file from the CLASSPATH by <exclusions> tag

on spring-boot-starter-jdbc dependency as show below

Go to dependency hierarchy tab ----> right click hikari cp jar file ---> select exclusion ---> ... ...

The above action reflects in pom.xml as shown below

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-jdbc</artifactId>

<exclusions>

<exclusion>

<groupId>com.zaxxer</groupId> <artifactId>HikariCP</artifactId>

</exclusion>

</exclusions>

</dependency>

step2) add Tomcat CP jar file to the CLASSPATH as dependnecy in pom.xml file

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->

<dependency>

<groupId>org.apache.tomcat</groupId> <artifactId>tomcat-jdbc</artifactId>

</dependency>

·)

step3) Run the client App

Popular DataSources are

a) HikariCP

b) Apache DBCp2

c) C3PO

d) Tomcat cp

e) Proxool CP

All these are standalone jdbc con pool

f) Broune CP g) Oracle UCP and etc..

To enable apache DBCP2 DataSource in Spring boot App

==========

==========

step1) make sure that hikaricp, tomcat cp jar files are removed from CLASSPATH using pom.xml file

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

<exclusions>

<exclusion>

<groupId>com.zaxxer</groupId>

<artifactId>HikariCP</artifactId>

</exclusion>

</exclusions> </dependency>

note:: The HikariCP DataSource is always good to use becoz it gives

are great performance.. but we learning other datasources configuration purely for the knowledge sake

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->

<!--<dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</dependency>-->

step2) add apache dbcp2 jar file...

note:: spring-boot-starter-jdbc jar file gives only hikaricp jar file as relavent jar file the remaining data sources related jar files will not come as relavent jar files.. So we need to add them manually..

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->

<dependency>

<groupId>org.apache.commons</groupId> <artifactId>commons-dbcp2</artifactId>

</dependency>

To enable oracle UCP DataSource in spring boot App

==========

==========

step1) make sure that hikaricp, tomcat cp, apache dbcp2 jar files are not added /removed from classpath

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

<exclusions>

<exclusion>

<groupId>com.zaxxer</groupId> <artifactId>HikariCP</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->

<!--<dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</dependency>-->

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->

<!--<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-dbcp2</artifactId>

</dependency>-->

step2) add oracle UCP jar file to the classpath using pom.xml

<!-- https://mvnrepository.com/artifact/com.oracle.database.jdbc/ucp -->

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ucp</artifactId>

</dependency>

if we place all the 4 datasources related depndencies at a time in spring boot App then what happens?

Ans) As per Algorithm priority the Hikari CP Data Source will be taken for AutoConfiguration.

DataSource algorithm in spring boot 1.x

a) Tomcat cp

b) hikari cp

c) apache dbcp2

d) apche dbcp

=> The Indurstry standard DataSource having for JDBC con pool is HikariDataSource (becoz of its performance)

note::: JDBC API can not be used to interact NO SQL DB s/ws.. It is purely given to interact with SQL DB s/ws.

How does the @EnableAutoConfigurtaion annotation of @SpringBootApplication knows which classes of the currently added jar files should configured as spring beans through auto Configuration?

Ans) The @EnableAutoConfiguation annotation searches for spring.factories file in META-INF folders of all the jar files added to the CLASSPATH but finds in spring-boot-autoconfigure-<ver>.jar file. This file is internally linked with another file called META-INF\spring

org.springframework.boot.autoconfigure.Auto Configuration.imports

This file contains all the Configuration classes that needs to executed

for Autoconfiguration activitity (nearly 144 classes are avaiable)

this chanages version to version (in 3.2.4 version 152)

All these configuration classes and thier nested and imported configuration classes related @Bean methods execute to get the objects of java classses as spring beans through AutoConfiuration activity

sample class names

org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration
org.springframework.boot.autoconfigure.jdbc.JndiDataSourceAutoConfiguration
org.springframework.boot.autoconfigure.jdbc.XADataSourceAutoConfiguration

@Import({ DataSourceConfiguration.Hikari.class, DataSourceConfiguration.Tomcat.class,

@AutoConfiguration(after = DataSourceAutoConfiguration.class) @ConditionalOnClass({ DataSource.class, JdbcTemplate.class

@ConditionalOnSingleCandidate(DataSource.class) @EnableConfigurationProperties(JdbcProperties.class)
@Import({ DatabaseInitialization DependencyConfigurer.class, } JdbcTemplateConfiguration.class,

DataSourceConfiguration.Dbcp2.class, DataSourceConfiguration.OracleUcp.class, DataSourceConfiguration.Generic.class, DataSourceJmxConfiguration.class })

protected static class PooledDataSourceConfiguration {

NamedParameterJdbcTemplateConfiguration.class })

public class JdbcTemplateAutoConfiguration {

}

@Bean

@Configuration Properties(prefix = "spring.datasource.hikari")

HikariDataSource dataSource(DataSource Properties properties) {

HikariDataSource dataSource = createDataSource(properties, HikariDataSource.class);

if (StringUtils.hasText(properties.getName())) {

@Configuration(proxyBeanMethods = false)

@ConditionalOnMissingBean(JdbcOperations.class)

class JdbcTemplateConfiguration {

@Bean

dataSource.setPoolName(properties.getName());

}

return dataSource;

}

@Primary

JdbcTemplate jdbcTemplate(DataSource dataSource, JdbcProperties properties) {

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

JdbcProperties.Template template = properties.getTemplate();

jdbcTemplate.setFetchSize(template.getFetchSize());

jdbcTemplate.setMaxRows(template.getMaxRows());

if (template.getQueryTimeout() != null) {

jdbcTemplate.setQueryTimeout((int) template.getQueryTimeout().getSeconds());

}

return jdbcTemplate;

}

file

Short anwser for interview ---> @EnableAutoconfigration searches for spring.factories in META-INF folder

from

of all jar files and gets spring-boot-autoconfiguration-<ver>.jar file .. This gets all the @Configuration classes to participate

in AutoConfiguation from the file META-INF\spring org.springframework.boot.autoconfigure. AutoConfiguration.imports and executes

all the @Bean methods of given Configuration class and thier nested, imported Configuration classes. These @Bean method returned

called

objs become spring beans automatically (this is auto configuraiton of of spring beans)

=======maven vedios=========

https://www.youtube.com/watch?v=2ZWK87ws2tM

https://www.youtube.com/watch?v=g8tyXC0fCd8

https://www.youtube.com/watch?v=OplE9qRhXl8

https://www.youtube.com/watch?v=WccmV8_MeC8

To get All bean ids maintained by the IOC container

===

3.x

String beanids[]=ctx.getBeanDefinitionNames();

org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration $PooledDataSourceConfiguration, jdbcConnectionDetails,

org.springframework.boot.autoconfigure.jdbc.metadata.DataSourcePoolMetadataProvidersConfiguration, org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,

System.out.println("All bean ids are ::"+Arrays.toString(beanids));

All bean ids are ::[org.springframework.context.annotation.internalConfigurationAnnotation Processor, org.springframework.context.annotation.internalAutowiredAnnotationProcessor, org.springframework.context.annotation.internalCommonAnnotation Processor, org.springframework.context.event.internalEventListener Processor, org.springframework.context.event.internalEventListenerFactory, bootProj04RealtimeDiMiniProjectLayeredApplication, org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory, empController, empDAO, empService, org.springframework.boot.autoconfigure. AutoConfiguration Packages, org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration, propertySources

PlaceholderConfigurer,
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration$ClassProxyingConfiguration,
forceAutoProxyCreator ToUseClassProxying,
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,
org.springframework.boot.autoconfigure.availability.ApplicationAvailabilityAutoConfiguration,
applicationAvailability, org.springframework.boot.autoconfigure.jdbc.DataSourceConfiguration$Hikari,
jdbcConnectionDetailsHikariBeanPostProcessor, dataSource,
org.springframework.boot.autoconfigure.jdbc.DataSourceJmxConfiguration$Hikari,
org.springframework.boot.autoconfigure.jdbc.DataSourceJmxConfiguration,
org.springframework.boot.autoconfigure.jdbc.metadata.DataSourcePoolMetadataProvidersConfiguration$Hi
kariPoolDataSourceMetadataProviderConfiguration, hikariPoolDataSourceMetadataProvider,

org.springframework.boot.context.properties.Configuration PropertiesBindingPostProcessor,
org.springframework.boot.context.internalConfiguration PropertiesBinder,

org.springframework.boot.context.properties.BoundConfiguration Properties,
org.springframework.boot.context.properties.EnableConfiguration Properties
Registrar.methodValidationExcludeFilter, spring.datasource-
org.springframework.boot.autoconfigure.jdbc.DataSource Properties,
org.springframework.boot.autoconfigure.transaction.TransactionManagerCustomizationAutoConfiguration,
platformTransactionManagerCustomizers, transactionExecutionListeners,
spring.transaction-org.springframework.boot.autoconfigure.transaction.TransactionProperties,
org.springframework.boot.autoconfigure.context.Configuration PropertiesAutoConfiguration,

org.springframework.boot.autoconfigure.context.LifecycleAutoConfiguration, lifecycleProcessor,
spring.lifecycle-org.springframework.boot.autoconfigure.context.LifecycleProperties,
org.springframework.boot.autoconfigure.dao.PersistenceException TranslationAutoConfiguration,
persistenceException TranslationPostProcessor,
org.springframework.boot.autoconfigure.info.ProjectInfoAutoConfiguration,
spring.info-org.springframework.boot.autoconfigure.info.ProjectInfoProperties,
org.springframework.boot.autoconfigure.jdbc.JdbcTemplateConfiguration, jdbcTemplate,
org.springframework.boot.autoconfigure.jdbc.NamedParameterJdbcTemplateConfiguration, named
ParameterJdbcTemplate, org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration,
spring.jdbc- org.springframework.boot.autoconfigure.jdbc.JdbcProperties,
org.springframework.boot.sql.init.dependency.DatabaseInitialization DependencyConfigurer$Depends
OnDatabaseInitializationPostProcessor,

org.springframework.boot.autoconfigure.jdbc.JdbcClientAutoConfiguration, jdbcClient,
org.springframework.boot.autoconfigure.sql.init.DataSourceInitializationConfiguration,
dataSourceScriptDatabaseInitializer,
org.springframework.boot.autoconfigure.sql.init.SqlInitializationAutoConfiguration,
spring.sql.init-org.springframework.boot.autoconfigure.sql.init.SqlInitializationProperties,

org.springframework.boot.autoconfigure.ssl.SslAutoConfiguration, fileWatcher,
sslPropertiesSslBundleRegistrar, sslBundleRegistry,
spring.ssl-org.springframework.boot.autoconfigure.ssl.SslProperties,
org.springframework.boot.autoconfigure.task.TaskExecutorConfigurations$TaskExecutorBuilderConfiguratio
n, taskExecutorBuilder, org.springframework.boot.autoconfigure.task.TaskExecutorConfigurations

org.springframework.boot.autoconfigure.task.TaskExecutorConfigurations$ThreadPoolTaskExecutorBuilderConfigurat
ion, threadPoolTaskExecutorBuilder,

$SimpleAsyncTaskExecutor BuilderConfiguration, simpleAsyncTaskExecutorBuilder,
org.springframework.boot.autoconfigure.task.TaskExecutorConfigurations$TaskExecutorConfiguration,
application TaskExecutor, org.springframework.boot.autoconfigure.task.TaskExecutionAutoConfiguration,

spring.task.execution-org.springframework.boot.autoconfigure.task.TaskExecutionProperties, org.springframework.boot.autoconfigure.task.TaskSchedulingConfigurations$ThreadPoolTaskSchedulerBuilderConfiguration, thread PoolTaskSchedulerBuilder, org.springframework.boot.autoconfigure.task.TaskSchedulingConfigurations$TaskSchedulerBuilderConfiguration, taskSchedulerBuilder, org.springframework.boot.autoconfigure.task.TaskSchedulingConfigurations$SimpleAsyncTaskSchedulerBuilderConfiguration, simpleAsyncTaskScheduler Builder, org.springframework.boot.autoconfigure.task.TaskSchedulingAutoConfiguration, spring.task.scheduling-org.springframework.boot.autoconfigure.task.TaskSchedulingProperties, org.springframework.boot.autoconfigure.jdbc.DataSource TransactionManagerAutoConfiguration$JdbcTransactionManagerConfiguration, transactionManager, org.springframework.boot.autoconfigure.jdbc.DataSource TransactionManagerAutoConfiguration, org.springframework.transaction.annotation. Proxy Transaction ManagementConfiguration, org.springframework.transaction.config.internalTransactionAdvisor, transactionAttributeSource, transactionInterceptor, org.springframework.transaction.config.internal TransactionalEventListenerFactory, org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration$Enable TransactionManagementConfiguration$CglibAutoProxyConfiguration, org.springframework.aop.config.internalAutoProxyCreator, org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration$EnableTransaction ManagementConfiguration, org.springframework.boot.autoconfigure.transaction. TransactionAutoConfiguration $Transaction TemplateConfiguration, transaction Template, org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration]

Spring 2.x DataSource Prioirity algorithm is

a) Hikari CP (default)

b) Tocmat cp

c) Apache DBCp2

d) Oracle UCP

Spring = frmaework

spring boot =spring ++ (framework of frmework)

How to break the above algorithm and how can we use certain DataSource directly with

out following the above algorithm?

While using these Data

 # Specify DataSource type

spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource

// for tomcat cp

(or)

Source types.. we need to add the relavent jar file to classpath (pom.xml)

spring.datasource.type=oracle.ucp.jdbc.PoolDataSourceImpl // for oracle UCP
spring.datasource.type=com.mchange.v2.c3p0.Combo Pooled DataSource

Popular standalone Datasources (Con pools)

HikariCP

Tomcat cp apache dbcp2 oracle UCP apache DBCP C3PO Proxool vibur brouneCp and etc..

Part of spring boot 2.x/3.x

DataSource algorithm

c3PO dependency in pom.xml file

<!-- https://mvnrepository.com/artifact/com.mchange/c3p0-->

<dependency>

<groupId>com.mchange</groupId>

we

Here we can specify any DataSources class name which is part of DataSource added in Autoconfiguration (hikaricp, Tomcat cp, apache dbcp2, oracle ucp) .or which is not part of DataSource algorithm (like c3p0, proxool, vibur and etc..)

How can make spring boot App working other than DataSource algorithm DataSoruces/Connection pool?

(or)

How can we use c3p0, proxool, vibur and etc.. jdbc con pools in spring boot application?

Ans1) Configure the related DataSource class as spring bean using @Bean method

in @SpringBootApplication class step1) add ur choice jdbc con pool dependency/jar file to the classpath

<!-- https://mvnrepository.com/artifact/c3p0/c3p0--> <dependency>

spring boot 3.x

The DataSource classname of any vendor supplied jdbc con pool can be gathered through its documents (search in google)

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

<artifactId>c3p0</artifactId>

<version>0.9.5.5</version>

</dependency>

</dependency>

step2) configure the relevent DataSource class name as as spring bean

in main class cum configuration class (@SpringBootApplication class) using @Ben method

```
@Bean(name="c3p0DS")
public Combo Pooled DataSource createC3PODS() throws Exception {
cpds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
System.out.println("BootProj03 LayeredAppApplication.createC3PODS()");
ComboPooledDataSource cpds=new Combo Pooled DataSource();
cpds.setDriverClass("oracle.jdbc.driver.Oracle Driver");
cpds.setUser("system"); cpds.setPassword("manager");
cpds.setInitialPoolSize(10); cpds.setMaxPoolSize(100);
return cpds;
}
```

if we keep the jar files of DataSource algorithm in classpath..then also

this @Bean method supplied DataSource will dominate/override the Datasource given by Algorithm

(or) Collecting jdbc properites from application.properties

application.properties

#DataSource cfg

spring.datasource.driver-class-name-oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=manager

c3p0.minsize=10

c3p0.maxsize=1000

user-defined keys

fixed keys

In main class

@Autowired

private Environment env;

@Bean(name="c3p0DS")

I

The properties file content, system properites content,

env.. variables content will be stored automatically

int "Environment" obj.. so we are injecting that obj to

our main class which is also a Spring Bean

public ComboPooledDataSource createC3PODS() throws Exception {

}

System.out.println("BootProj03 LayeredAppApplication.createC3PODS()");

ComboPooledDataSource cpds=new Combo Pooled DataSource();

cpds.setDriverClass(env.getRequiredProperty("spring.datasource.driver-class-name"));

cpds.setJdbcUrl(env.getRequiredProperty("spring.datasource.url'"));

cpds.setUser(env.getRequiredProperty("spring.datasource.username"));

cpds.setPassword(env.getRequiredProperty("spring.datasource.password"));

cpds.setInitialPoolSize(Integer.parseInt(env.getRequiredProperty("c3p0.minsize")));

cpds.setMaxPoolSize(Integer.parseInt(env.getRequiredProperty("c3p0.maxsize")));

return cpds;

(or)

Ans2) specify the relavent new DataSource/con pool related DataSource class name in (Best)
application.properties as the value of "spring.datasource.type" key

application.properties

spring.datasource.type=com.mchange.v2.c3p0.Combo Pooled DataSource

(for working with c3PO jdbc con pool)

is

(becoz it easy)

is

Q) When spring-boot-starter-jdbc is added to CLASSPATH what default DataSource we get?

Ans) HikariCP DataSource

Q) if @Bean method of main class (@SBA class) giving non- DataSource algorithm DataSource obj

which

then the Spring Boot App takes DataSoruce objecti

?

Ans) @Bean method generated DataSource object will override /dominate the DataSource alogithm's DataSoruce

(or) spring.datasource.type property DataSoruce

(if both are there @Bean will get

Q) Which is industry standard DataSource to use in real projects?

the higher priority)

Ans) Hikari CP as of now becoz its performence (especially speed of creating jdbc con objects in the jdbc con pool)

Q) how to sepcify min pool size and max pool and othe additional properties while working with HikariCp Jdbc con pool?

we can addtional hikaricp specific properties as shown below in application.properties file

application.properties

#jdbc properties

spring.datasource.driver-class-name-oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=tiger

spring.datasource.hikari.maximum-pool-size=100

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.keepalive-time=100000

we can find this kind of "Specific DataSource type properties" for hikaricp, tomcat cp, apache dbcp2 and oracle ucp

For apache dbcp2

spring.datasource.dbcp2.abandoned-usage-tracking

C

spring.datasource.dbcp2.access-to-underlying-connection-allowed

a

For hikari CP

spring.datasource.dbcp2.auto-commit-on-return

spring.datasource.hikari.allow-pool-suspension

spring.datasource.dbcp2.cache-state

spring.datasource.hikari.auto-commit

spring.datasource.dbcp2.clear-statement-pool-on-return

spring.datasource.hikari.catalog

spring.datasource.dbcp2.connection-factory-class-name

spring.datasource.hikari.connection-init-sql

spring.datasource.dbcp2.connection-init-sqls

spring.datasource.hikari.connection-test-query

spring.datasource.dbcp2.default-auto-commit

spring.datasource.hikari.connection-timeout

spring.datasource.dbcp2.default-catalog

spring.datasource.hikari.data-source-class-name

spring.datasource.dbcp2.default-query-timeout

spring.datasource.hikari.data-source-j-n-d-i

spring.datasource.dbcp2.default-read-only

spring.datasource.hikari.data-source-properties

spring.datasource.dbcp2.default-schema

spring.datasource.hikari.driver-class-name

spring.datasource.dbcp2.default-transaction-isolation

spring.datasource.hikari.exception-override-class-name

spring.datasource.dbcp2.disconnection-sql-codes

spring.datasource.hikari.health-check-properties

spring.datasource.dbcp2.driver

spring.datasource.hikari.idle-timeout

spring.datasource.dbcp2.driver-class-name

spring.datasource.hikari.initialization-fail-timeout

spring.datasource.dbcp2.eviction-policy-class-name

spring.datasource.hikari.isolate-internal-queries

spring.datasource.dbcp2.fast-fail-validation

spring.datasource.hikari.jdbc-url

spring.datasource.dbcp2.initial-size

spring.datasource.hikari.keepalive-time

spring.datasource.dbcp2.jmx-name

spring.datasource.hikari.leak-detection-threshold

spring.datasource.hikari.login-timeout

spring.datasource.hikari.max-lifetime

spring.datasource.hikari.maximum-pool-size

#For Oracle UCP

spring.datasource.oracleucp.abandoned-connection-timeout

ucpl

spring.datasource.oracleucp.connection-factory-class-name

spring.datasource.oracleucp.connection-factory-properties

spring.datasource.oracleucp.connection-harvest-max-count

spring.datasource.oracleucp.connection-harvest-trigger-count

spring.datasource.oracleucp.connection-labeling-high-cost

spring.datasource.oracleucp.connection-pool-name

spring.datasource.oracleucp.connection-properties

spring.datasource.oracleucp.connection-repurpose-threshold

spring.datasource.oracleucp.connection-validation-timeout

spring.datasource.oracleucp.connection-wait-timeout

spring.datasource.oracleucp.data-source-name

spring.datasource.oracleucp.database-name

spring.datasource.oracleucp.description

spring.datasource.oracleucp.fast-connection-failover-enabled

spring.datasource.oracleucp.high-cost-connection-reuse-threshold

spring.datasource.oracleucp.inactive-connection-timeout

spring.datasource.oracleucp.initial-pool-size

spring.datasource.oracleucp.login-timeout

spring.datasource.oracleucp.max-connection-reuse-count

spring.datasource.oracleucp.max-connection-reuse-time

spring.datasource.oracleucp.max-connections-per-shard

spring.datasource.oracleucp.max-idle-time

How can

#For Tomcat CP spring.datasource.tomcat.abandon-when-percentage-full

spring.datasource.tomcat.access-to-underlying-connection-allowed

spring.datasource.tomcat.alternate-username-allowed

spring.datasource.tomcat.commit-on-return

spring.datasource.tomcat.connection-properties

spring.datasource.tomcat.data-source-j-n-d-i

spring.datasource.tomcat.db-properties

spring.datasource.tomcat.default-auto-commit

spring.datasource.tomcat.default-catalog

spring.datasource.tomcat.default-read-only

spring.datasource.tomcat.default-transaction-isolation

spring.datasource.tomcat.driver-class-name

spring.datasource.tomcat.fair-queue

spring.datasource.tomcat.ignore-exception-on-pre-load

spring.datasource.tomcat.init-s-q-1

spring.datasource.tomcat.initial-size

spring.datasource.tomcat.jdbc-interceptors

spring.datasource.tomcat.jmx-enabled

https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties.data

of

we make certain java class starter not become as spring bean as part of AutoConfiguration Process?

eg: when we add spring-boot-starter-jdbc we get multiple classes as spring beans through AutoConfiguration process, the classes are

a) HikariDataSource

b)JdbcTemplate

c)NamedParameterJdbcTemplate

d) DriverManagerDataSource

and etc,.

note:: if we feel we do not need these spring beans as part of AutoConfiguration process then we need to take exclude then in AutoConfigration process using the exclude attribute of @SpringBootApplication annotation

In main class

==========

```
@SpringBootApplication (exclude = {JdbcTemplateAutoConfiguration.class})
public class BootProj04EmployeeSearchAppLayeredApplication {
...
}
```

This will exclude JdbcTemlate,Named ParameterJdbcTemplate

classes being participating in AutoConfiguration activity

we can collect AutoConfiguration class names

from

META-INF

spring

aot.factories

=>In spring, spring boot f/ws the modules jdbc, ORM, data jpa are given to locate and interact with SQL DB s/w

=>In spring, Spring boot f/ws for interacting with every NO SQL Db s/w we have separate module

spring data mongodb, spring data cassendra, spring data neo4j and etc..

=>The Spring Boot Autoconfiguration Feature makes only pre-defined classes that are available in

the starters added to the CLASSPATH as spring beans.. In any angle it does not make user-defined

classes annotated with streo type annotations as the spring beans (user-defined classes are scanned becoz of @ComponentScan annotation)

org.springframework.boot.autoconfigure.AutoConfiguration.imports

of spring-boot-autoconfigure-<ver>.jar file