

# Day\_1(Spring Boot)

15 December 2023 12:02 PM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.trining.app</groupId>
  <artifactId>HelloWorldSpringBoot</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <java.version>1.14</java.version>
  </properties>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.4.RELEASE</version>
  </parent>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>

    </dependency>
  </dependencies>

</project>
```

□The spring boot annotations are mostly placed in org.springframework.boot.autoconfigure and org.springframework.boot.autoconfigure.condition packages. Let's learn about some frequently used spring boot annotations as well as which work behind the scene.

## @SpringBootApplication

Spring boot is mostly about auto-configuration. This auto-configuration is done by component scanning i.e. finding all classes in classpath for @Component annotation. It also involve scanning of @Configuration annotation and initialize some extra beans.

@SpringBootApplication annotation enable all able things in one step. It enables the three features:

@EnableAutoConfiguration : enable auto-configuration mechanism

@ComponentScan : enable @Component scan

@SpringBootConfiguration : register extra beans in the context

@GetMapping: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

@PostMapping: It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

@PutMapping: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

@DeleteMapping: It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

@PatchMapping: It maps the HTTP PATCH requests on the specific handler method. It is used instead of using: @RequestMapping(method = RequestMethod.PATCH)

@RequestBody: It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.

@ResponseBody: It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

@PathVariable: It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

@RequestParam: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

@RequestHeader: It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

@RestController: It can be considered as a combination of @Controller and @ResponseBody annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.

@ModelAttribute: It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @ModelAttribute annotation, we can access objects that are populated on the server-side.

```
package com.spring.org;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

public class Greeting {

    private final long id;
    private final String content;

    public Greeting(long id, String content) {
        this.id = id;
        this.content = content;
    }

    public long getId() {
        return id;
    }

    public String getContent() {
        return content;
    }

    //      @Override
    //      public String toString() {
    //          return "Greeting [id=" + id + ", content=" + content +
    "]" +
    //      }
}
```

```
package com.spring.org;

public class Employee {
    private int id;
    private String firstName;
    private String lastName;
    private String email;

    public Employee() {
        super();
    }
}
```

```

    }
    public Employee(int id, String firstName, String lastName, String email)
    {
        super();
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    @Override
    public String toString() {
        return "Employee [id=" + id + ", firstName=" + firstName + ", lastName="
        + lastName + ", email=" + email + "]";
    }
}

```

```

package com.spring.org;

import java.util.ArrayList;
import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController

```

```

public class HelloWorld {
//      @GetMapping("/")
//      public String getMessage() {
//          return "<h1>HelloWorld application</h1>";
//      }

@GetMapping("/")
public String getMessage() {
return "<form action=\"http://localhost:8080/greeting\"
method=\"get\">\r\n"
    "\r\n"
    "\r\n"
    "        Enter your message <input type=\"text\" name=\"name\"> <br>
    <input type=\"submit\" name=\"\" value=\"Submit\">\r\n"
    "\r\n"
    "</form>"
    }

@GetMapping("/greeting")
public Greeting greetingMessage(@RequestParam(value = "name",
defaultValue = "World") String name) {
return new Greeting(1, "WELCOME");
}

//      @GetMapping("/greeting")
//      public Greeting greetingMessage(@RequestParam(value = "name",
//          defaultValue = "World") String name) {
//          return new Greeting(1,name);
//      }

@GetMapping("/employee")
public List<Employee> getEmployee() {
//      emptyEmployee new ArrayList<Employee>();
employeeList.add(new Employee(1, "Eshan", "Kashyap", "e@gmail.com"));
employeeList.add(new Employee(2, "Esha", "Kashyap", "e@gmail.com"));
employeeList.add(new Employee(1, "Eshan", "Kashyap", "e@gmail.com"));

return employeeList;
}
}

```

```

package com.spring.org;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

```

```
//@RestController
@SpringBootApplication
public class HelloSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
    /*
        * @GetMapping("/") public String getMessage() { return
        * "Hello spring Application";
        *
        * }
        */
}
```

## Day2\_(commandLineRunner)

18 December 2023 12:40 PM

How to use CommandLineRunner

You can use CommandLineRunner interface in three ways:

- 1) Using CommandLineRunner as @Component
- 2) Implement CommandLineRunner in @SpringBootApplication
- 3) Using CommandLineRunner as Bean

```
package com.training.org.SpringApplication;

import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class Application implements CommandLineRunner{
    Logger logger = LoggerFactory.getLogger(Application.class);

    @Autowired
    ApplicationContext applicationContext;

    public static void main(String[] args) {
        System.out.println("Main Method called");
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {

        Map<String, Object> beansOfType = applicationContext.getBeansOfType(Object.class);
        beansOfType.forEach((s, o) -> {
            logger.info("{} - {}", s, o.getClass());
        });
        System.out.println("Run method is called");
        logger.info("This is logging info");
        logger.warn("This is warning message");
        logger.error("this is error message");

    }

}

package com.training.org.SpringApplication;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class ApplicationRunner implements CommandLineRunner{

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Run method from command line runner ");
    }

}
```

```

package com.training.org.SpringApplication;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component
@Order(value = 2)
public class ApplicationRunner implements CommandLineRunner{

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Run method from command line runner ");
    }
}

package com.training.org.SpringApplication;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
@Component
@Order(value = 1)
public class ApplicationRunner2 implements CommandLineRunner{
    @Override
    public void run(String... args) throws Exception {
        System.out.println("2 Run method from command line runner ");
    }
}

package com.training.org.SpringApplication;

import org.springframework.boot.CommandLineRunner;

public class NewApplicationRunner implements CommandLineRunner {
    @Override
    public void run(String...args) throws Exception {
        System.out.println(" New Application runner ApplicationRunner called");
    }
}

package com.training.org.SpringApplication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application{

    public static void main(String[] args) {
        System.out.println("Main Method called");
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public NewApplicationRunner applicationStartupRunner() {
        return new NewApplicationRunner();
    }
}

```

```
}
```

```
package com.training.org.SpringApplication;

import org.springframework.stereotype.Component;

@Component
public class City {

    private int id;

    private String name;
    private int population;

    public City() {
        System.out.println("Default constructor of City is called");
    }

    public City(String name, int population) {
        System.out.println("Parameterised constructor of City is called");
        this.name = name;
        this.population = population;
    }

    public int getId() {
        return id;
    }

    public void setId(int i) {
        this.id = i;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPopulation() {
        return population;
    }

    public void setPopulation(int population) {
        this.population = population;
    }

    @Override
    public String toString() {

        var builder = new StringBuilder();
        builder.append("City{id=").append(id).append(", name=")
            .append(name).append(", population=")
            .append(population).append("}");

        return builder.toString();
    }
}
```

```
package com.training.org.SpringApplication;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;

public class NewApplicationRunner implements CommandLineRunner {

    @Autowired
```



```

        City city;

@Override
public void run(String...args) throws Exception {

    city.setId(1);
    city.setName("Pune");
    city.setPopulation(1000000);

    System.out.println(" New Application runner ApplicationRunner called");
}
}

```

2nd project:

```

package com.example.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class LoggingApplication {
    Logger logger = LoggerFactory.getLogger(LoggingApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(LoggingApplication.class, args);
    }

    @GetMapping("/")
    public String getLogMessage() {
        logger.info("This is logger information");
        logger.warn("This is warning message");
        logger.error("This is error message");
        return "Check your log message";
    }
}

logging.level.root=ERROR
logging.file.name=logs/mylog.log
spring.application.name= MyApplication

```

Let's see how to include a Logback configuration with a different color and logging pattern, with separate specifications for console and file output, and with a decent rolling policy to avoid generating huge log files.  
First of all, we should go toward a solution which allows handling our logging settings alone, instead of polluting application.properties, which is commonly used for many other application settings.

When a file in the classpath has one of the following names, Spring Boot will automatically load it over the default configuration:

```

logback-spring.xml
logback.xml
logback-spring.groovy
logback.groovy

```

# Pattern:

```
logging.pattern.console=%d{yy-MMMM-dd HH:mm:ss:SSS} %5p %t %c{2}:%L - %m%n
```

# Output:

```
18-January-17 02:21:20:317 WARN http-nio-8080-exec-1 o.o.s.c.MainController:22 - This is WARN
18-January-17 02:21:20:320 ERROR http-nio-8080-exec-1 o.o.s.c.MainController:23 - This is ERROR
```

# Pattern:

```
logging.pattern.console= %d{dd/MM/yyyy HH:mm:ss.SSS} %-5level [%thread] %logger{115} - %msg%n
```

# Output:

```
17/01/2018 02:15:15.052 WARN [http-nio-8080-exec-1] org.o7planning.sblogging.controller.MainController - This is WARN
17/01/2018 02:15:15.054 ERROR [http-nio-8080-exec-1] org.o7planning.sblogging.controller.MainController - This is ERROR
```

And you can change "Logging pattern" by customizing the following properties:

logging.pattern.console

logging.pattern.file

# Pattern:

```
logging.pattern.console= %d{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{15} - %msg%n
```

# Output:

```
2018-Jan-17 01:58:49.958 WARN [http-nio-8080-exec-1] o.o.s.c.MainController - This is WARN
2018-Jan-17 01:58:49.960 ERROR [http-nio-8080-exec-1] o.o.s.c.MainController - This is ERROR
```

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>
        %d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n
      </Pattern>
    </layout>
  </appender>

  <root level="info">
    <appender-ref ref="Console" />
  </root>

  <!-- Log everything at the TRACE level -->
  <logger name="com.training.org" level="trace" additivity="false">
    <appender-ref ref="Console" />
  </logger>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>
        [%thread] %d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n
      </Pattern>
    </layout>
  </appender>

  <root level="info">
    <appender-ref ref="Console" />
  </root>
```

```
<!-- Log everything at the TRACE level -->  
<logger name="com.example.demo" level="warn" additivity="false">  
  <appender-ref ref="Console" />  
</logger>  
</configuration>
```

# Properties(Core Properties)

18 December 2023 01:00 PM

There are sixteen categories of Spring Boot Property are as follows:

- Core Properties
- Cache Properties
- Mail Properties
- JSON Properties
- Data Properties
- Transaction Properties
- Data Migration Properties
- Integration Properties
- Web Properties
- Templating Properties
- Server Properties
- Security Properties
- RSocket Properties
- Actuator Properties
- DevTools Properties
- Testing Properties

```
# =====
# COMMON SPRING BOOT PROPERTIES
#
# This sample file is provided as a guideline. Do NOT copy it in its
# entirety to your own application.      ^^^
# =====

# -----
# CORE PROPERTIES
# -----
debug=false # Enable debug logs.
trace=false # Enable trace logs.

# LOGGING
logging.config= # Location of the logging configuration file. For instance, 'classpath:logback.xml' for Logback.
logging.exception-conversion-word=%wEx # Conversion word used when logging exceptions.
logging.file= # Log file name (for instance, 'myapp.log'). Names can be an exact location or relative to the current directory.
logging.file.max-history=0 # Maximum of archive log files to keep. Only supported with the default logback setup.
logging.file.max-size=10MB # Maximum log file size. Only supported with the default logback setup.
logging.level.*= # Log levels severity mapping. For instance, 'logging.level.org.springframework=DEBUG'.
logging.path= # Location of the log file. For instance, '/var/log'.
logging.pattern.console= # Appender pattern for output to the console. Supported only with the default Logback setup.
logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS # Appender pattern for log date format. Supported only with the default Logback setup.
logging.pattern.file= # Appender pattern for output to a file. Supported only with the default Logback setup.
logging.pattern.level=%5p # Appender pattern for log level. Supported only with the default Logback setup.
logging.register-shutdown-hook=false # Register a shutdown hook for the logging system when it is initialized.

# AOP
spring.aop.auto=true # Add @EnableAspectJAutoProxy.
spring.aop.proxy-target-class=true # Whether subclass-based (CGLIB) proxies are to be created (true), as opposed to standard Java interface-based proxies (false).

# IDENTITY (ContextIdApplicationContextInitializer)
spring.application.name= # Application name.

# ADMIN (SpringApplicationAdminJmxAutoConfiguration)
spring.application.admin.enabled=false # Whether to enable admin features for the application.
spring.application.admin.jmx-name=org.springframework.boot:type=Admin,name=SpringApplication # JMX name of the application admin MBean.

# AUTO-CONFIGURATION
spring.autoconfigure.exclude= # Auto-configuration classes to exclude.

# BANNER
spring.banner.charset=UTF-8 # Banner file encoding.
spring.banner.location=classpath:banner.txt # Banner text resource location.
spring.banner.image.location=classpath:banner.gif # Banner image file location (jpg or png can also be used).
spring.banner.image.width=76 # Width of the banner image in chars.
spring.banner.image.height= # Height of the banner image in chars (default based on image height).
spring.banner.image.margin=2 # Left hand image margin in chars.
spring.banner.image.invert=false # Whether images should be inverted for dark terminal themes.

# SPRING CORE
spring.beaninfo.ignore=true # Whether to skip search of BeanInfo classes.

# SPRING CACHE (CacheProperties)
spring.cache.cache-names= # Comma-separated list of cache names to create if supported by the underlying cache manager.
spring.cache.caffeine.spec= # The spec to use to create caches. See CaffeineSpec for more details on the spec format.
spring.cache.couchbase.expiration=0ms # Entry expiration. By default the entries never expire. Note that this value is ultimately converted to seconds.
spring.cache.ehcache.config= # The location of the configuration file to use to initialize EhCache.
spring.cache.infinispan.config= # The location of the configuration file to use to initialize Infinispan.
spring.cache.jcache.config= # The location of the configuration file to use to initialize the cache manager.
```

spring.cache.jcache.provider= # Fully qualified name of the CachingProvider implementation to use to retrieve the JSR-107 compliant cache manager. Needed only if more than one JSR-107 implementation is available on the classpath.  
 spring.cache.redis.cache-null-values=true # Allow caching null values.  
 spring.cache.redis.key-prefix= # Key prefix.  
 spring.cache.redis.time-to-live=0ms # Entry expiration. By default the entries never expire.  
 spring.cache.redis.use-key-prefix=true # Whether to use the key prefix when writing to Redis.  
 spring.cache.type= # Cache type. By default, auto-detected according to the environment.

# SPRING CONFIG - using environment property only (ConfigFileApplicationListener)  
 spring.config.additional-location= # Config file locations used in addition to the defaults.  
 spring.config.location= # Config file locations that replace the defaults.  
 spring.config.name=application # Config file name.

# HAZELCAST (HazelcastProperties)  
 spring.hazelcast.config= # The location of the configuration file to use to initialize Hazelcast.

# PROJECT INFORMATION (ProjectInfoProperties)  
 spring.info.build.location=classpath:META-INF/build-info.properties # Location of the generated build-info.properties file.  
 spring.info.git.location=classpath:git.properties # Location of the generated git.properties file.

# JMX  
 spring.jmx.default-domain= # JMX domain name.  
 spring.jmx.enabled=true # Expose management beans to the JMX domain.  
 spring.jmx.server=mbeanServer # MBeanServer bean name.

# Email (MailProperties)  
 spring.mail.default-encoding=UTF-8 # Default MimeMessage encoding.  
 spring.mail.host= # SMTP server host. For instance, `smtp.example.com`.  
 spring.mail.jndi-name= # Session JNDI name. When set, takes precedence over other Session settings.  
 spring.mail.password= # Login password of the SMTP server.  
 spring.mail.port= # SMTP server port.  
 spring.mail.properties.\*= # Additional JavaMail Session properties.  
 spring.mail.protocol=smtp # Protocol used by the SMTP server.  
 spring.mail.test-connection=false # Whether to test that the mail server is available on startup.  
 spring.mail.username= # Login user of the SMTP server.

# APPLICATION SETTINGS (SpringApplication)  
 spring.main.banner-mode=console # Mode used to display the banner when the application runs.  
 spring.main.sources= # Sources (class names, package names, or XML resource locations) to include in the ApplicationContext.  
 spring.main.web-application-type= # Flag to explicitly request a specific type of web application. If not set, auto-detected based on the classpath.

# FILE ENCODING (FileEncodingApplicationListener)  
 spring.mandatory-file-encoding= # Expected character encoding the application must use.

# INTERNATIONALIZATION (MessageSourceProperties)  
 spring.messages.always-use-message-format=false # Whether to always apply the MessageFormat rules, parsing even messages without arguments.  
 spring.messages.basename=messages # Comma-separated list of basenames (essentially a fully-qualified classpath location), each following the ResourceBundle convention with relaxed support for slash based locations.  
 spring.messages.cache-duration= # Loaded resource bundle files cache duration. When not set, bundles are cached forever. If a duration suffix is not specified, seconds will be used.  
 spring.messages.encoding=UTF-8 # Message bundles encoding.  
 spring.messages.fallback-to-system-locale=true # Whether to fall back to the system Locale if no files for a specific Locale have been found.  
 spring.messages.use-code-as-default-message=false # Whether to use the message code as the default message instead of throwing a "NoSuchMessageException". Recommended during development only.

# OUTPUT  
 spring.output.ansi.enabled=detect # Configures the ANSI output.

# PID FILE (ApplicationPidFileWriter)  
 spring.pid.fail-on-write-error= # Fails if ApplicationPidFileWriter is used but it cannot write the PID file.  
 spring.pid.file= # Location of the PID file to write (if ApplicationPidFileWriter is used).

# PROFILES  
 spring.profiles.active= # Comma-separated list of active profiles. Can be overridden by a command line switch.  
 spring.profiles.include= # Unconditionally activate the specified comma-separated list of profiles (or list of profiles if using YAML).

# QUARTZ SCHEDULER (QuartzProperties)  
 spring.quartz.jdbc.comment-prefix=-- # Prefix for single-line comments in SQL initialization scripts.  
 spring.quartz.jdbc.initialize-schema=embedded # Database schema initialization mode.  
 spring.quartz.jdbc.schema=classpath:org/quartz/impl/jdbcjobstore/tables\_@.sql # Path to the SQL file to use to initialize the database schema.  
 spring.quartz.job-store-type=memory # Quartz job store type.  
 spring.quartz.properties.\*= # Additional Quartz Scheduler properties.

# REACTOR (ReactorCoreProperties)  
 spring.reactor.stacktrace-mode.enabled=false # Whether Reactor should collect stacktrace information at runtime.

# SENDGRID (SendGridAutoConfiguration)  
 spring.sendgrid.api-key= # SendGrid API key.  
 spring.sendgrid.proxy.host= # SendGrid proxy host.  
 spring.sendgrid.proxy.port= # SendGrid proxy port.

# Properties(Web Properties)

18 December 2023 01:02 PM

```
# -----
# WEB PROPERTIES
# -----

# EMBEDDED SERVER CONFIGURATION (ServerProperties)
server.address= # Network address to which the server should bind.
server.compression.enabled=false # Whether response compression is enabled.
server.compression.excluded-user-agents= # List of user-agents to exclude from compression.
server.compression.mime-types=text/html,text/xml,text/plain,text/css,text/javascript,application/javascript # Comma-separated list of MIME types that
should be compressed.
server.compression.min-response-size=2048 # Minimum "Content-Length" value that is required for compression to be performed.
server.connection-timeout= # Time that connectors wait for another HTTP request before closing the connection. When not set, the connector's container-
specific default is used. Use a value of -1 to indicate no (that is, an infinite) timeout.
server.error.include-exception=false # Include the "exception" attribute.
server.error.include-stacktrace=never # When to include a "stacktrace" attribute.
server.error.path=/error # Path of the error controller.
server.error.whitelabel.enabled=true # Whether to enable the default error page displayed in browsers in case of a server error.
server.http2.enabled=false # Whether to enable HTTP/2 support, if the current environment supports it.
server.jetty.acceptors= # Number of acceptor threads to use.
server.jetty.accesslog.append=false # Append to log.
server.jetty.accesslog.date-format=dd/MMM/yyyy:HH:mm:ss Z # Timestamp format of the request log.
server.jetty.accesslog.enabled=false # Enable access log.
server.jetty.accesslog.extended-format=false # Enable extended NCSA format.
server.jetty.accesslog.file-date-format= # Date format to place in log file name.
server.jetty.accesslog.filename= # Log filename. If not specified, logs redirect to "System.err".
server.jetty.accesslog.locale= # Locale of the request log.
server.jetty.accesslog.log-cookies=false # Enable logging of the request cookies.
server.jetty.accesslog.log-latency=false # Enable logging of request processing time.
server.jetty.accesslog.log-server=false # Enable logging of the request hostname.
server.jetty.accesslog.retention-period=31 # Number of days before rotated log files are deleted.
server.jetty.accesslog.time-zone=GMT # Timezone of the request log.
server.jetty.max-http-post-size=0 # Maximum size, in bytes, of the HTTP post or put content.
server.jetty.selectors= # Number of selector threads to use.
server.max-http-header-size=0 # Maximum size, in bytes, of the HTTP message header.
server.port=8080 # Server HTTP port.
server.server-header= # Value to use for the Server response header (if empty, no header is sent).
server.use-forward-headers= # Whether X-Forwarded-* headers should be applied to the HttpRequest.
server.servlet.context-parameters.*= # Servlet context init parameters.
server.servlet.context-path= # Context path of the application.
server.servlet.application-display-name=application # Display name of the application.
server.servlet.jsp.class-name=org.apache.jasper.servlet.JspServlet # The class name of the JSP servlet.
server.servlet.jsp.init-parameters.*= # Init parameters used to configure the JSP servlet.
server.servlet.jsp.registered=true # Whether the JSP servlet is registered.
server.servlet.path=/ # Path of the main dispatcher servlet.
server.servlet.session.cookie.comment= # Comment for the session cookie.
server.servlet.session.cookie.domain= # Domain for the session cookie.
server.servlet.session.cookie.http-only= # "HttpOnly" flag for the session cookie.
server.servlet.session.cookie.max-age= # Maximum age of the session cookie. If a duration suffix is not specified, seconds will be used.
server.servlet.session.cookie.name= # Session cookie name.
server.servlet.session.cookie.path= # Path of the session cookie.
server.servlet.session.cookie.secure= # "Secure" flag for the session cookie.
server.servlet.session.persistent=false # Whether to persist session data between restarts.
server.servlet.session.store-dir= # Directory used to store session data.
server.servlet.session.timeout= # Session timeout. If a duration suffix is not specified, seconds will be used.
server.servlet.session.tracking-modes= # Session tracking modes (one or more of the following: "cookie", "url", "ssl").
server.ssl.ciphers= # Supported SSL ciphers.
server.ssl.client-auth= # Whether client authentication is wanted ("want") or needed ("need"). Requires a trust store.
server.ssl.enabled= # Enable SSL support.
server.ssl.enabled-protocols= # Enabled SSL protocols.
server.ssl.key-alias= # Alias that identifies the key in the key store.
server.ssl.key-password= # Password used to access the key in the key store.
server.ssl.key-store= # Path to the key store that holds the SSL certificate (typically a jks file).
server.ssl.key-store-password= # Password used to access the key store.
server.ssl.key-store-provider= # Provider for the key store.
server.ssl.key-store-type= # Type of the key store.
server.ssl.protocol=TLS # SSL protocol to use.
server.ssl.trust-store= # Trust store that holds SSL certificates.
server.ssl.trust-store-password= # Password used to access the trust store.
server.ssl.trust-store-provider= # Provider for the trust store.
server.ssl.trust-store-type= # Type of the trust store.
server.tomcat.accept-count=0 # Maximum queue length for incoming connection requests when all possible request processing threads are in use.
server.tomcat.accesslog.buffered=true # Whether to buffer output such that it is flushed only periodically.
```

```

server.tomcat.accesslog.directory=logs # Directory in which log files are created. Can be absolute or relative to the Tomcat base dir.
server.tomcat.accesslog.enabled=false # Enable access log.
server.tomcat.accesslog.file-date-format=.yyyy-MM-dd # Date format to place in the log file name.
server.tomcat.accesslog.pattern=common # Format pattern for access logs.
server.tomcat.accesslog.prefix=access_log # Log file name prefix.
server.tomcat.accesslog.rename-on-rotate=false # Whether to defer inclusion of the date stamp in the file name until rotate time.
server.tomcat.accesslog.request-attributes-enabled=false # Set request attributes for the IP address, Hostname, protocol, and port used for the request.
server.tomcat.accesslog.rotate=true # Whether to enable access log rotation.
server.tomcat.accesslog.suffix=.log # Log file name suffix.
server.tomcat.additional-tld-skip-patterns= # Comma-separated list of additional patterns that match jars to ignore for TLD scanning.
server.tomcat.background-processor-delay=30s # Delay between the invocation of backgroundProcess methods. If a duration suffix is not specified, seconds will be used.
server.tomcat.basedir= # Tomcat base directory. If not specified, a temporary directory is used.
server.tomcat.internal-proxies=10\.\d{1,3}\.\d{1,3}\.\d{1,3}\|\\
    192\.\.168\.\d{1,3}\.\d{1,3}\|\\
    169\.\.254\.\d{1,3}\.\d{1,3}\|\\
    127\.\.\d{1,3}\.\d{1,3}\.\d{1,3}\|\\
    172\.\.1[6-9]{1}\.\d{1,3}\.\d{1,3}\|\\
    172\.\.2[0-9]{1}\.\d{1,3}\.\d{1,3}\|\\
    172\.\.3[0-1]{1}\.\d{1,3}\.\d{1,3} # Regular expression matching trusted IP addresses.
server.tomcat.max-connections=0 # Maximum number of connections that the server accepts and processes at any given time.
server.tomcat.max-http-header-size=0 # Maximum size, in bytes, of the HTTP message header.
server.tomcat.max-http-post-size=0 # Maximum size, in bytes, of the HTTP post content.
server.tomcat.max-threads=0 # Maximum number of worker threads.
server.tomcat.min-spare-threads=0 # Minimum number of worker threads.
server.tomcat.port-header=X-Forwarded-Port # Name of the HTTP header used to override the original port value.
server.tomcat.protocol-header= # Header that holds the incoming protocol, usually named "X-Forwarded-Proto".
server.tomcat.protocol-header-https-value=https # Value of the protocol header indicating whether the incoming request uses SSL.
server.tomcat.redirect-context-root= # Whether requests to the context root should be redirected by appending a / to the path.
server.tomcat.remote-ip-header= # Name of the HTTP header from which the remote IP is extracted. For instance, 'X-FORWARDED-FOR'.
server.tomcat.resource.cache-ttl= # Time-to-live of the static resource cache.
server.tomcat.uri-encoding=UTF-8 # Character encoding to use to decode the URI.
server.tomcat.use-relative-redirects= # Whether HTTP 1.1 and later location headers generated by a call to sendRedirect will use relative or absolute redirects.
server.undertow.accesslog.dir= # Undertow access log directory.
server.undertow.accesslog.enabled=false # Whether to enable the access log.
server.undertow.accesslog.pattern=common # Format pattern for access logs.
server.undertow.accesslog.prefix=access_log. # Log file name prefix.
server.undertow.accesslog.rotate=true # Whether to enable access log rotation.
server.undertow.accesslog.suffix=log # Log file name suffix.
server.undertow.buffer-size= # Size of each buffer, in bytes.
server.undertow.direct-buffers= # Whether to allocate buffers outside the Java heap.
server.undertow.io-threads= # Number of I/O threads to create for the worker.
server.undertow.eager-filter-init=true # Whether servlet filters should be initialized on startup.
server.undertow.max-http-post-size=0 # Maximum size, in bytes, of the HTTP post content.
server.undertow.worker-threads= # Number of worker threads.

# FREEMARKER (FreeMarkerProperties)
spring.freemarker.allow-request-override=false # Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.
spring.freemarker.allow-session-override=false # Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.
spring.freemarker.cache=false # Whether to enable template caching.
spring.freemarker.charset=UTF-8 # Template encoding.
spring.freemarker.check-template-location=true # Whether to check that the templates location exists.
spring.freemarker.content-type=text/html # Content-Type value.
spring.freemarker.enabled=true # Whether to enable MVC view resolution for this technology.
spring.freemarker.expose-request-attributes=false # Whether all request attributes should be added to the model prior to merging with the template.
spring.freemarker.expose-session-attributes=false # Whether all HttpSession attributes should be added to the model prior to merging with the template.
spring.freemarker.expose-spring-macro-helpers=true # Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".
spring.freemarker.prefer-file-system-access=true # Whether to prefer file system access for template loading. File system access enables hot detection of template changes.
spring.freemarker.prefix= # Prefix that gets prepended to view names when building a URL.
spring.freemarker.request-context-attribute= # Name of the RequestContext attribute for all views.
spring.freemarker.settings.*= # Well-known FreeMarker keys which are passed to FreeMarker's Configuration.
spring.freemarker.suffix=.ftl # Suffix that gets appended to view names when building a URL.
spring.freemarker.template-loader-path=classpath:/templates/ # Comma-separated list of template paths.
spring.freemarker.view-names= # White list of view names that can be resolved.

# GROOVY TEMPLATES (GroovyTemplateProperties)
spring.groovy.template.allow-request-override=false # Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.
spring.groovy.template.allow-session-override=false # Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.
spring.groovy.template.cache=false # Whether to enable template caching.
spring.groovy.template.charset=UTF-8 # Template encoding.
spring.groovy.template.check-template-location=true # Whether to check that the templates location exists.

```

```

spring.groovy.template.configuration.*= # See GroovyMarkupConfigurer
spring.groovy.template.content-type=text/html # Content-Type value.
spring.groovy.template.enabled=true # Whether to enable MVC view resolution for this technology.
spring.groovy.template.expose-request-attributes=false # Whether all request attributes should be added to the model prior to merging with the template.
spring.groovy.template.expose-session-attributes=false # Whether all HttpSession attributes should be added to the model prior to merging with the template.
spring.groovy.template.expose-spring-macro-helpers=true # Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".
spring.groovy.template.prefix= # Prefix that gets prepended to view names when building a URL.
spring.groovy.template.request-context-attribute= # Name of the RequestContext attribute for all views.
spring.groovy.template.resource-loader-path=classpath:/templates/ # Template path.
spring.groovy.template.suffix=.tpl # Suffix that gets appended to view names when building a URL.
spring.groovy.template.view-names= # White list of view names that can be resolved.

# SPRING HATEOAS (HateoasProperties)
spring.hateoas.use-hal-as-default-json-media-type=true # Whether application/hal+json responses should be sent to requests that accept application/json.

# HTTP message conversion
spring.http.converters.preferred-json-mapper= # Preferred JSON mapper to use for HTTP message conversion. By default, auto-detected according to the environment.

# HTTP encoding (HttpEncodingProperties)
spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and responses. Added to the "Content-Type" header if not set explicitly.
spring.http.encoding.enabled=true # Whether to enable http encoding support.
spring.http.encoding.force= # Whether to force the encoding to the configured charset on HTTP requests and responses.
spring.http.encoding.force-request= # Whether to force the encoding to the configured charset on HTTP requests. Defaults to true when "force" has not been specified.
spring.http.encoding.force-response= # Whether to force the encoding to the configured charset on HTTP responses.
spring.http.encoding.mapping= # Locale in which to encode mapping.

# MULTIPART (MultipartProperties)
spring.servlet.multipart.enabled=true # Whether to enable support of multipart uploads.
spring.servlet.multipart.file-size-threshold=0 # Threshold after which files are written to disk. Values can use the suffixes "MB" or "KB" to indicate megabytes or kilobytes, respectively.
spring.servlet.multipart.location= # Intermediate location of uploaded files.
spring.servlet.multipart.max-file-size=1MB # Max file size. Values can use the suffixes "MB" or "KB" to indicate megabytes or kilobytes, respectively.
spring.servlet.multipart.max-request-size=10MB # Max request size. Values can use the suffixes "MB" or "KB" to indicate megabytes or kilobytes, respectively.
spring.servlet.multipart.resolve-lazily=false # Whether to resolve the multipart request lazily at the time of file or parameter access.

# JACKSON (JacksonProperties)
spring.jackson.date-format= # Date format string or a fully-qualified date format class name. For instance, `yyyy-MM-dd HH:mm:ss`.
spring.jackson.default-property-inclusion= # Controls the inclusion of properties during serialization. Configured with one of the values in Jackson's JsonInclude.Include enumeration.
spring.jackson.deserialization.*= # Jackson on/off features that affect the way Java objects are deserialized.
spring.jackson.generator.*= # Jackson on/off features for generators.
spring.jackson.joda-date-time-format= # Joda date time format string. If not configured, "date-format" is used as a fallback if it is configured with a format string.
spring.jackson.locale= # Locale used for formatting.
spring.jackson.mapper.*= # Jackson general purpose on/off features.
spring.jackson.parser.*= # Jackson on/off features for parsers.
spring.jackson.property-naming-strategy= # One of the constants on Jackson's PropertyNamingStrategy. Can also be a fully-qualified class name of a PropertyNamingStrategy subclass.
spring.jackson.serialization.*= # Jackson on/off features that affect the way Java objects are serialized.
spring.jackson.time-zone= # Time zone used when formatting dates. For instance, "America/Los_Angeles" or "GMT+10".

# GSON (GsonProperties)
spring.gson.date-format= # Format to use when serializing Date objects.
spring.gson.disable-html-escaping= # Whether to disable the escaping of HTML characters such as '<', '>', etc.
spring.gson.disable-inner-class-serialization= # Whether to exclude inner classes during serialization.
spring.gson.enable-complex-map-key-serialization= # Whether to enable serialization of complex map keys (i.e. non-primitives).
spring.gson.exclude-fields-without-expose-annotation= # Whether to exclude all fields from consideration for serialization or deserialization that do not have the "Expose" annotation.
spring.gson.field-naming-policy= # Naming policy that should be applied to an object's field during serialization and deserialization.
spring.gson.generate-non-executable-json= # Whether to generate non executable JSON by prefixing the output with some special text.
spring.gson.lenient= # Whether to be lenient about parsing JSON that doesn't conform to RFC 4627.
spring.gson.long-serialization-policy= # Serialization policy for Long and long types.
spring.gson.pretty-printing= # Whether to output serialized JSON that fits in a page for pretty printing.
spring.gson.serialize-nulls= # Whether to serialize null fields.

# JERSEY (JerseyProperties)
spring.jersey.application-path= # Path that serves as the base URI for the application. If specified, overrides the value of "@ApplicationPath".
spring.jersey.filter.order=0 # Jersey filter chain order.
spring.jersey.init.*= # Init parameters to pass to Jersey through the servlet or filter.
spring.jersey.servlet.load-on-startup=-1 # Load on startup priority of the Jersey servlet.
spring.jersey.type=servlet # Jersey integration type.

# SPRING LDAP (LdapProperties)

```



```

spring ldap.anonymous-read-only=false # Whether read-only operations should use an anonymous environment.
spring ldap.base= # Base suffix from which all operations should originate.
spring ldap.base-environment.*= # LDAP specification settings.
spring ldap.password= # Login password of the server.
spring ldap.urls= # LDAP URLs of the server.
spring ldap.username= # Login username of the server.

# EMBEDDED LDAP (EmbeddedLdapProperties)
spring ldap.embedded.base-dn= # List of base DN's.
spring ldap.embedded.credential.username= # Embedded LDAP username.
spring ldap.embedded.credential.password= # Embedded LDAP password.
spring ldap.embedded.ldif=classpath:schema.ldif # Schema (LDIF) script resource reference.
spring ldap.embedded.port=0 # Embedded LDAP port.
spring ldap.embedded.validation.enabled=true # Whether to enable LDAP schema validation.
spring ldap.embedded.validation.schema= # Path to the custom schema.

# MUSTACHE TEMPLATES (MustacheAutoConfiguration)
spring mustache.allow-request-override=false # Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.
spring mustache.allow-session-override=false # Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.
spring mustache.cache=false # Whether to enable template caching.
spring mustache.charset=UTF-8 # Template encoding.
spring mustache.check-template-location=true # Whether to check that the templates location exists.
spring mustache.content-type=text/html # Content-Type value.
spring mustache.enabled=true # Whether to enable MVC view resolution for this technology.
spring mustache.expose-request-attributes=false # Whether all request attributes should be added to the model prior to merging with the template.
spring mustache.expose-session-attributes=false # Whether all HttpSession attributes should be added to the model prior to merging with the template.
spring mustache.expose-spring-macro-helpers=true # Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".
spring mustache.prefix=classpath:/templates/ # Prefix to apply to template names.
spring mustache.request-context-attribute= # Name of the RequestContext attribute for all views.
spring mustache.suffix=.mustache # Suffix to apply to template names.
spring mustache.view-names= # White list of view names that can be resolved.

# SPRING MVC (WebMvcProperties)
spring.mvc.async.request-timeout= # Amount of time before asynchronous request handling times out.
spring.mvc.contentnegotiation.favor-parameter=false # Whether a request parameter ("format" by default) should be used to determine the requested media type.
spring.mvc.contentnegotiation.favor-path-extension=false # Whether the path extension in the URL path should be used to determine the requested media type.
spring.mvc.contentnegotiation.media-types.*= # Map file extensions to media types for content negotiation. For instance, yaml to text/yaml.
spring.mvc.contentnegotiation.parameter-name= # Query parameter name to use when "favor-parameter" is enabled.
spring.mvc.date-format= # Date format to use. For instance, 'dd/MM/yyyy'.
spring.mvc.dispatch-trace-request=false # Whether to dispatch TRACE requests to the FrameworkServlet doService method.
spring.mvc.dispatch-options-request=true # Whether to dispatch OPTIONS requests to the FrameworkServlet doService method.
spring.mvc.favicon.enabled=true # Whether to enable resolution of favicon.ico.
spring.mvc.formcontent.putfilter.enabled=true # Whether to enable Spring's HttpPutFormContentFilter.
spring.mvc.ignore-default-model-on-redirect=true # Whether the content of the "default" model should be ignored during redirect scenarios.
spring.mvc.locale= # Locale to use. By default, this locale is overridden by the "Accept-Language" header.
spring.mvc.locale-resolver=accept-header # Define how the locale should be resolved.
spring.mvc.log-resolved-exception=false # Whether to enable warn logging of exceptions resolved by a "HandlerExceptionResolver".
spring.mvc.message-codes-resolver-format= # Formatting strategy for message codes. For instance, 'PREFIX_ERROR_CODE'.
spring.mvc.pathmatch.use-registered-suffix-pattern=false # Whether suffix pattern matching should work only against extensions registered with "spring.mvc.contentnegotiation.media-types.*".
spring.mvc.pathmatch.use-suffix-pattern=false # Whether to use suffix pattern match (".*") when matching patterns to requests.
spring.mvc.servlet.load-on-startup=-1 # Load on startup priority of the dispatcher servlet.
spring.mvc.static-path-pattern=/** # Path pattern used for static resources.
spring.mvc.throw-exception-if-no-handler-found=false # Whether a "NoHandlerFoundException" should be thrown if no Handler was found to process a request.
spring.mvc.view.prefix= # Spring MVC view prefix.
spring.mvc.view.suffix= # Spring MVC view suffix.

# SPRING RESOURCES HANDLING (ResourceProperties)
spring.resources.add-mappings=true # Whether to enable default resource handling.
spring.resources.cache.cachecontrol.cache-private= # Indicate that the response message is intended for a single user and must not be stored by a shared cache.
spring.resources.cache.cachecontrol.cache-public= # Indicate that any cache may store the response.
spring.resources.cache.cachecontrol.max-age= # Maximum time the response should be cached, in seconds if no duration suffix is not specified.
spring.resources.cache.cachecontrol.must-revalidate= # Indicate that once it has become stale, a cache must not use the response without re-validating it with the server.
spring.resources.cache.cachecontrol.no-cache= # Indicate that the cached response can be reused only if re-validated with the server.
spring.resources.cache.cachecontrol.no-store= # Indicate to not cache the response in any case.
spring.resources.cache.cachecontrol.no-transform= # Indicate intermediaries (caches and others) that they should not transform the response content.
spring.resources.cache.cachecontrol.proxy-revalidate= # Same meaning as the "must-revalidate" directive, except that it does not apply to private caches.
spring.resources.cache.cachecontrol.s-max-age= # Maximum time the response should be cached by shared caches, in seconds if no duration suffix is not specified.
spring.resources.cache.cachecontrol.stale-if-error= # Maximum time the response may be used when errors are encountered, in seconds if no duration suffix

```

is not specified.

spring.resources.cache.cachecontrol.stale-while-revalidate= # Maximum time the response can be served after it becomes stale, in seconds if no duration suffix is not specified.  
spring.resources.cache.period= # Cache period for the resources served by the resource handler. If a duration suffix is not specified, seconds will be used.  
spring.resources.chain.cache=true # Whether to enable caching in the Resource chain.  
spring.resources.chain.enabled= # Whether to enable the Spring Resource Handling chain. By default, disabled unless at least one strategy has been enabled.  
spring.resources.chain.gziped=false # Whether to enable resolution of already gzipped resources.  
spring.resources.chain.html-application-cache=false # Whether to enable HTML5 application cache manifest rewriting.  
spring.resources.chain.strategy.content.enabled=false # Whether to enable the content Version Strategy.  
spring.resources.chain.strategy.content.paths=/\*\* # Comma-separated list of patterns to apply to the content Version Strategy.  
spring.resources.chain.strategy.fixed.enabled=false # Whether to enable the fixed Version Strategy.  
spring.resources.chain.strategy.fixed.paths=/\*\* # Comma-separated list of patterns to apply to the fixed Version Strategy.  
spring.resources.chain.strategy.fixed.version= # Version string to use for the fixed Version Strategy.  
spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/ # Locations of static resources.

#### # SPRING SESSION (SessionProperties)

spring.session.store-type= # Session store type.  
spring.session.timeout= # Session timeout. If a duration suffix is not specified, seconds will be used.  
spring.session.servlet.filter-order=-2147483598 # Session repository filter order.  
spring.session.servlet.filter-dispatcher-types=async,error,request # Session repository filter dispatcher types.

#### # SPRING SESSION HAZELCAST (HazelcastSessionProperties)

spring.session.hazelcast.flush-mode=on-save # Sessions flush mode.  
spring.session.hazelcast.map-name=spring:session:sessions # Name of the map used to store sessions.

#### # SPRING SESSION JDBC (JdbcSessionProperties)

spring.session.jdbc.cleanup-cron=0 \* \* \* \* \* # Cron expression for expired session cleanup job.  
spring.session.jdbc.initialize-schema=embedded # Database schema initialization mode.  
spring.session.jdbc.schema=classpath:org/springframework/session/jdbc/schema-@.sql # Path to the SQL file to use to initialize the database schema.  
spring.session.jdbc.table-name=SPRING\_SESSION # Name of the database table used to store sessions.

#### # SPRING SESSION MONGODB (MongoSessionProperties)

spring.session.mongodb.collection-name=sessions # Collection name used to store sessions.

#### # SPRING SESSION REDIS (RedisSessionProperties)

spring.session.redis.cleanup-cron=0 \* \* \* \* \* # Cron expression for expired session cleanup job.  
spring.session.redis.flush-mode=on-save # Sessions flush mode.  
spring.session.redis.namespace=spring:session # Namespace for keys used to store sessions.

#### # THYMELEAF (ThymeleafAutoConfiguration)

spring.thymeleaf.cache=true # Whether to enable template caching.  
spring.thymeleaf.check-template=true # Whether to check that the template exists before rendering it.  
spring.thymeleaf.check-template-location=true # Whether to check that the templates location exists.  
spring.thymeleaf.enabled=true # Whether to enable Thymeleaf view resolution for Web frameworks.  
spring.thymeleaf.enable-spring-el-compiler=false # Enable the SpringEL compiler in SpringEL expressions.  
spring.thymeleaf.encoding=UTF-8 # Template files encoding.  
spring.thymeleaf.excluded-view-names= # Comma-separated list of view names (patterns allowed) that should be excluded from resolution.  
spring.thymeleaf.mode=HTML # Template mode to be applied to templates. See also Thymeleaf's TemplateMode enum.  
spring.thymeleaf.prefix=classpath:/templates/ # Prefix that gets prepended to view names when building a URL.  
spring.thymeleaf.reactive.chunked-mode-view-names= # Comma-separated list of view names (patterns allowed) that should be the only ones executed in CHUNKED mode when a max chunk size is set.  
spring.thymeleaf.reactive.full-mode-view-names= # Comma-separated list of view names (patterns allowed) that should be executed in FULL mode even if a max chunk size is set.  
spring.thymeleaf.reactive.max-chunk-size=0 # Maximum size of data buffers used for writing to the response, in bytes.  
spring.thymeleaf.reactive.media-types= # Media types supported by the view technology.  
spring.thymeleaf.servlet.content-type=text/html # Content-Type value written to HTTP responses.  
spring.thymeleaf.suffix=.html # Suffix that gets appended to view names when building a URL.  
spring.thymeleaf.template-resolver-order= # Order of the template resolver in the chain.  
spring.thymeleaf.view-names= # Comma-separated list of view names (patterns allowed) that can be resolved.

#### # SPRING WEBFLUX (WebFluxProperties)

spring.webflux.date-format= # Date format to use. For instance, 'dd/MM/yyyy'.  
spring.webflux.static-path-pattern=/\*\* # Path pattern used for static resources.

#### # SPRING WEB SERVICES (WebServicesProperties)

spring.webservices.path=/services # Path that serves as the base URI for the services.  
spring.webservices.servlet.init= # Servlet init parameters to pass to Spring Web Services.  
spring.webservices.servlet.load-on-startup=-1 # Load on startup priority of the Spring Web Services servlet.  
spring.webservices.wsdl-locations= # Comma-separated list of locations of WSDLs and accompanying XSDs to be exposed as beans.

# Properties (Security Properties)

18 December 2023 01:04 PM

```
# -----  
# SECURITY PROPERTIES  
# -----  
# SECURITY (SecurityProperties)  
spring.security.filter.order=-100 # Security filter chain order.  
spring.security.filter.dispatcher-types=async,error,request # Security filter chain dispatcher types.  
spring.security.user.name=user # Default user name.  
spring.security.user.password= # Password for the default user name.  
spring.security.user.roles= # Granted roles for the default user name.  
  
# SECURITY OAUTH2 CLIENT (OAuth2ClientProperties)  
spring.security.oauth2.client.provider.*= # OAuth provider details.  
spring.security.oauth2.client.registration.*= # OAuth client registrations.
```

# Properties(Data Properties)

18 December 2023 01:04 PM

```
# -----
# DATA PROPERTIES
# -----

# FLYWAY (FlywayProperties)
spring.flyway.baseline-description= #
spring.flyway.baseline-on-migrate= #
spring.flyway.baseline-version=1 # Version to start migration
spring.flyway.check-location=true # Whether to check that migration scripts location exists.
spring.flyway.clean-disabled= #
spring.flyway.clean-on-validation-error= #
spring.flyway.dry-run-output= #
spring.flyway.enabled=true # Whether to enable flyway.
spring.flyway.encoding= #
spring.flyway.error-handlers= #
spring.flyway.group= #
spring.flyway.ignore-future-migrations= #
spring.flyway.ignore-missing-migrations= #
spring.flyway.init-sqls= # SQL statements to execute to initialize a connection immediately after obtaining it.
spring.flyway.installed-by= #
spring.flyway.locations=classpath:db/migration # The locations of migrations scripts.
spring.flyway.mixed= #
spring.flyway.out-of-order= #
spring.flyway.password= # JDBC password to use if you want Flyway to create its own DataSource.
spring.flyway.placeholder-prefix= #
spring.flyway.placeholder-replacement= #
spring.flyway.placeholder-suffix= #
spring.flyway.placeholders.*= #
spring.flyway.repeatable-sql-migration-prefix= #
spring.flyway.schemas= # schemas to update
spring.flyway.skip-default-callbacks= #
spring.flyway.skip-default-resolvers= #
spring.flyway.sql-migration-prefix=V #
spring.flyway.sql-migration-separator= #
spring.flyway.sql-migration-suffix=.sql #
spring.flyway.sql-migration-suffixes= #
spring.flyway.table= #
spring.flyway.target= #
spring.flyway.undo-sql-migration-prefix= #
spring.flyway.url= # JDBC url of the database to migrate. If not set, the primary configured data source is used.
spring.flyway.user= # Login user of the database to migrate.
spring.flyway.validate-on-migrate= #

# LIQUIBASE (LiquibaseProperties)
spring.liquibase.change-log-classpath:/db/changelog/db.changelog-master.yaml # Change log configuration path.
spring.liquibase.check-change-log-location=true # Whether to check that the change log location exists.
spring.liquibase.contexts= # Comma-separated list of runtime contexts to use.
spring.liquibase.default-schema= # Default database schema.
spring.liquibase.drop-first=false # Whether to first drop the database schema.
spring.liquibase.enabled=true # Whether to enable Liquibase support.
spring.liquibase.labels= # Comma-separated list of runtime labels to use.
spring.liquibase.parameters.*= # Change log parameters.
spring.liquibase.password= # Login password of the database to migrate.
spring.liquibase.rollback-file= # File to which rollback SQL is written when an update is performed.
spring.liquibase.url= # JDBC URL of the database to migrate. If not set, the primary configured data source is used.
spring.liquibase.user= # Login user of the database to migrate.

# COUCHBASE (CouchbaseProperties)
spring.couchbase.bootstrap-hosts= # Couchbase nodes (host or IP address) to bootstrap from.
spring.couchbase.bucket.name=default # Name of the bucket to connect to.
spring.couchbase.bucket.password= # Password of the bucket.
spring.couchbase.env.endpoints.key-value=1 # Number of sockets per node against the key/value service.
spring.couchbase.env.endpoints.queryservice.min-endpoints=1 # Minimum number of sockets per node.
spring.couchbase.env.endpoints.queryservice.max-endpoints=1 # Maximum number of sockets per node.
spring.couchbase.env.endpoints.viewservice.min-endpoints=1 # Minimum number of sockets per node.
spring.couchbase.env.endpoints.viewservice.max-endpoints=1 # Maximum number of sockets per node.
spring.couchbase.env.ssl.enabled= # Whether to enable SSL support. Enabled automatically if a "keyStore" is provided unless specified otherwise.
spring.couchbase.env.ssl.key-store= # Path to the JVM key store that holds the certificates.
spring.couchbase.env.ssl.key-store-password= # Password used to access the key store.
spring.couchbase.env.timeouts.connect=5000ms # Bucket connections timeouts.
spring.couchbase.env.timeouts.key-value=2500ms # Blocking operations performed on a specific key timeout.
spring.couchbase.env.timeouts.query=7500ms # N1QL query operations timeout.
spring.couchbase.env.timeouts.socket-connect=1000ms # Socket connect connections timeout.
spring.couchbase.env.timeouts.view=7500ms # Regular and geospatial view operations timeout.

# DAO (PersistenceExceptionTranslationAutoConfiguration)
spring.dao.exceptiontranslation.enabled=true # Whether to enable the PersistenceExceptionTranslationPostProcessor.

# CASSANDRA (CassandraProperties)
spring.data.cassandra.cluster-name= # Name of the Cassandra cluster.
```

spring.data.cassandra.compression=none # Compression supported by the Cassandra binary protocol.  
spring.data.cassandra.connect-timeout= # Socket option: connection time out.  
spring.data.cassandra.consistency-level= # Queries consistency level.  
spring.data.cassandra.contact-points=localhost # Cluster node addresses.  
spring.data.cassandra.fetch-size= # Queries default fetch size.  
spring.data.cassandra.keyspace-name= # Keyspace name to use.  
spring.data.cassandra.load-balancing-policy= # Class name of the load balancing policy.  
spring.data.cassandra.port= # Port of the Cassandra server.  
spring.data.cassandra.password= # Login password of the server.  
spring.data.cassandra.pool.heartbeat-interval=30s # Heartbeat interval after which a message is sent on an idle connection to make sure it's still alive. If a duration suffix is not specified, seconds will be used.  
spring.data.cassandra.pool.idle-timeout=120s # Idle timeout before an idle connection is removed. If a duration suffix is not specified, seconds will be used.  
spring.data.cassandra.pool.max-queue-size=256 # Maximum number of requests that get queued if no connection is available.  
spring.data.cassandra.pool.pool-timeout=5000ms # Pool timeout when trying to acquire a connection from a host's pool.  
spring.data.cassandra.read-timeout= # Socket option: read time out.  
spring.data.cassandra.reconnection-policy= # Reconnection policy class.  
spring.data.cassandra.repositories.type=auto # Type of Cassandra repositories to enable.  
spring.data.cassandra.retry-policy= # Class name of the retry policy.  
spring.data.cassandra.serial-consistency-level= # Queries serial consistency level.  
spring.data.cassandra.schema-action=none # Schema action to take at startup.  
spring.data.cassandra.ssl=false # Enable SSL support.  
spring.data.cassandra.username= # Login user of the server.

#### # DATA COUCHBASE (CouchbaseDataProperties)

spring.data.couchbase.auto-index=false # Automatically create views and indexes.  
spring.data.couchbase.consistency=read-your-own-writes # Consistency to apply by default on generated queries.  
spring.data.couchbase.repositories.type=auto # Type of Couchbase repositories to enable.

#### # ELASTICSEARCH (ElasticsearchProperties)

spring.data.elasticsearch.cluster-name=elasticsearch # Elasticsearch cluster name.  
spring.data.elasticsearch.cluster-nodes= # Comma-separated list of cluster node addresses.  
spring.data.elasticsearch.properties.\*= # Additional properties used to configure the client.  
spring.data.elasticsearch.repositories.enabled=true # Whether to enable Elasticsearch repositories.

#### # DATA LDAP

spring.data.ldap.repositories.enabled=true # Whether to enable LDAP repositories.

#### # MONGODB (MongoProperties)

spring.data.mongodb.authentication-database= # Authentication database name.  
spring.data.mongodb.database= # Database name.  
spring.data.mongodb.field-naming-strategy= # Fully qualified name of the FieldNamingStrategy to use.  
spring.data.mongodb.grid-fs-database= # GridFS database name.  
spring.data.mongodb.host= # Mongo server host. Cannot be set with URI.  
spring.data.mongodb.password= # Login password of the mongo server. Cannot be set with URI.  
spring.data.mongodb.port= # Mongo server port. Cannot be set with URI.  
spring.data.mongodb.repositories.type=auto # Type of Mongo repositories to enable.  
spring.data.mongodb.uri=mongodb://localhost/test # Mongo database URI. Cannot be set with host, port and credentials.  
spring.data.mongodb.username= # Login user of the mongo server. Cannot be set with URI.

#### # DATA REDIS

spring.data.redis.repositories.enabled=true # Whether to enable Redis repositories.

#### # NEO4J (Neo4jProperties)

spring.data.neo4j.auto-index=none # Auto index mode.  
spring.data.neo4j.embedded.enabled=true # Whether to enable embedded mode if the embedded driver is available.  
spring.data.neo4j.open-in-view=true # Register OpenSessionInViewInterceptor. Binds a Neo4j Session to the thread for the entire processing of the request.  
spring.data.neo4j.password= # Login password of the server.  
spring.data.neo4j.repositories.enabled=true # Whether to enable Neo4j repositories.  
spring.data.neo4j.uri= # URI used by the driver. Auto-detected by default.  
spring.data.neo4j.username= # Login user of the server.

#### # DATA REST (RepositoryRestProperties)

spring.data.rest.base-path= # Base path to be used by Spring Data REST to expose repository resources.  
spring.data.rest.default-media-type= # Content type to use as a default when none is specified.  
spring.data.rest.default-page-size= # Default size of pages.  
spring.data.rest.detection-strategy=default # Strategy to use to determine which repositories get exposed.  
spring.data.rest.enable-enum-translation= # Whether to enable enum value translation through the Spring Data REST default resource bundle.  
spring.data.rest.limit-param-name= # Name of the URL query string parameter that indicates how many results to return at once.  
spring.data.rest.max-page-size= # Maximum size of pages.  
spring.data.rest.page-param-name= # Name of the URL query string parameter that indicates what page to return.  
spring.data.rest.return-body-on-create= # Whether to return a response body after creating an entity.  
spring.data.rest.return-body-on-update= # Whether to return a response body after updating an entity.  
spring.data.rest.sort-param-name= # Name of the URL query string parameter that indicates what direction to sort results.

#### # SOLR (SolrProperties)

spring.data.solr.host=http://127.0.0.1:8983/solr # Solr host. Ignored if "zk-host" is set.  
spring.data.solr.repositories.enabled=true # Whether to enable Solr repositories.  
spring.data.solr.zk-host= # ZooKeeper host address in the form HOST:PORT.

#### # DATA WEB (SpringDataWebProperties)

spring.data.web.pageable.default-page-size=20 # Default page size.  
spring.data.web.pageable.max-page-size=2000 # Maximum page size to be accepted.  
spring.data.web.pageable.one-indexed-parameters=false # Whether to expose and assume 1-based page number indexes.  
spring.data.web.pageable.page-parameter=page # Page index parameter name.  
spring.data.web.pageable.prefix= # General prefix to be prepended to the page number and page size parameters.  
spring.data.web.pageable.qualifier-delimiter=\_ # Delimiter to be used between the qualifier and the actual page number and size properties.  
spring.data.web.pageable.size-parameter=size # Page size parameter name.  
spring.data.web.sort.sort-parameter=sort # Sort parameter name.

```

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.continue-on-error=false # Whether to stop if an error occurs while initializing the database.
spring.datasource.data= # Data (DML) script resource references.
spring.datasource.data-username= # Username of the database to execute DML scripts (if different).
spring.datasource.data-password= # Password of the database to execute DML scripts (if different).
spring.datasource.dbcp2.*= # Commons DBCP2 specific settings
spring.datasource.driver-class-name= # Fully qualified name of the JDBC driver. Auto-detected based on the URL by default.
spring.datasource.generate-unique-name=false # Whether to generate a random datasource name.
spring.datasource.hikari.*= # Hikari specific settings
spring.datasource.initialization-mode=embedded # Initialize the datasource with available DDL and DML scripts.
spring.datasource.jmx-enabled=false # Whether to enable JMX support (if provided by the underlying pool).
spring.datasource.jndi-name= # JNDI location of the datasource. Class, url, username & password are ignored when set.
spring.datasource.name= # Name of the datasource. Default to "testdb" when using an embedded database.
spring.datasource.password= # Login password of the database.
spring.datasource.platform=all # Platform to use in the DDL or DML scripts (such as schema-$(platform).sql or data-$(platform).sql).
spring.datasource.schema= # Schema (DDL) script resource references.
spring.datasource.schema-username= # Username of the database to execute DDL scripts (if different).
spring.datasource.schema-password= # Password of the database to execute DDL scripts (if different).
spring.datasource.separator=; # Statement separator in SQL initialization scripts.
spring.datasource.sql-script-encoding= # SQL scripts encoding.
spring.datasource.tomcat.*= # Tomcat datasource specific settings
spring.datasource.type= # Fully qualified name of the connection pool implementation to use. By default, it is auto-detected from the classpath.
spring.datasource.url= # JDBC URL of the database.
spring.datasource.username= # Login username of the database.
spring.datasource.xa.data-source-class-name= # XA datasource fully qualified name.
spring.datasource.xa.properties= # Properties to pass to the XA data source.

# JEST (Elasticsearch HTTP client) (JestProperties)
spring.elasticsearch.jest.connection-timeout=3s # Connection timeout.
spring.elasticsearch.jest.multi-threaded=true # Whether to enable connection requests from multiple execution threads.
spring.elasticsearch.jest.password= # Login password.
spring.elasticsearch.jest.proxy.host= # Proxy host the HTTP client should use.
spring.elasticsearch.jest.proxy.port= # Proxy port the HTTP client should use.
spring.elasticsearch.jest.read-timeout=3s # Read timeout.
spring.elasticsearch.jest.uris=http://localhost:9200 # Comma-separated list of the Elasticsearch instances to use.
spring.elasticsearch.jest.username= # Login username.

# H2 Web Console (H2ConsoleProperties)
spring.h2.console.enabled=false # Whether to enable the console.
spring.h2.console.path=/h2-console # Path at which the console is available.
spring.h2.console.settings.trace=false # Whether to enable trace output.
spring.h2.console.settings.web-allow-others=false # Whether to enable remote access.

# InfluxDB (InfluxDbProperties)
spring.influx.password= # Login password.
spring.influx.url= # URL of the InfluxDB instance to which to connect.
spring.influx.user= # Login user.

# JOOQ (JooqProperties)
spring.jooq.sql-dialect= # SQL dialect to use. Auto-detected by default.

# JDBC (JdbcProperties)
spring.jdbc.template.fetch-size=-1 # Number of rows that should be fetched from the database when more rows are needed.
spring.jdbc.template.max-rows=-1 # Maximum number of rows.
spring.jdbc.template.query-timeout= # Query timeout. Default is to use the JDBC driver's default configuration. If a duration suffix is not specified, seconds will be used.

# JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)
spring.data.jpa.repositories.enabled=true # Whether to enable JPA repositories.
spring.jpa.database= # Target database to operate on, auto-detected by default. Can be alternatively set using the "databasePlatform" property.
spring.jpa.database-platform= # Name of the target database to operate on, auto-detected by default. Can be alternatively set using the "Database" enum.
spring.jpa.generate-ddl=false # Whether to initialize the schema on startup.
spring.jpa.hibernate.ddl-auto= # DDL mode. This is actually a shortcut for the "hibernate.hbm2ddl.auto" property. Defaults to "create-drop" when using an embedded database and no schema manager was detected. Otherwise, defaults to "none".
spring.jpa.hibernate.naming.implicit-strategy= # Fully qualified name of the implicit naming strategy.
spring.jpa.hibernate.naming.physical-strategy= # Fully qualified name of the physical naming strategy.
spring.jpa.hibernate.use-new-id-generator-mappings= # Whether to use Hibernate's newer IdentifierGenerator for AUTO, TABLE and SEQUENCE.
spring.jpa.mapping-resources= # Mapping resources (equivalent to "mapping-file" entries in persistence.xml).
spring.jpa.open-in-view=true # Register OpenEntityManagerInViewInterceptor. Binds a JPA EntityManager to the thread for the entire processing of the request.
spring.jpa.properties.*= # Additional native properties to set on the JPA provider.
spring.jpa.show-sql=false # Whether to enable logging of SQL statements.

# JTA (JtaAutoConfiguration)
spring.jta.enabled=true # Whether to enable JTA support.
spring.jta.log-dir= # Transaction logs directory.
spring.jta.transaction-manager-id= # Transaction manager unique identifier.

# ATOMIKOS (AtomikosProperties)
spring.jta.atomikos.connectionfactory.borrow-connection-timeout=30 # Timeout, in seconds, for borrowing connections from the pool.
spring.jta.atomikos.connectionfactory.ignore-session-transacted-flag=true # Whether to ignore the transacted flag when creating session.
spring.jta.atomikos.connectionfactory.local-transaction-mode=false # Whether local transactions are desired.
spring.jta.atomikos.connectionfactory.maintenance-interval=60 # The time, in seconds, between runs of the pool's maintenance thread.
spring.jta.atomikos.connectionfactory.max-idle-time=60 # The time, in seconds, after which connections are cleaned up from the pool.
spring.jta.atomikos.connectionfactory.max-lifetime=0 # The time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit.
spring.jta.atomikos.connectionfactory.max-pool-size=1 # The maximum size of the pool.
spring.jta.atomikos.connectionfactory.min-pool-size=1 # The minimum size of the pool.
spring.jta.atomikos.connectionfactory.reap-timeout=0 # The reap timeout, in seconds, for borrowed connections. 0 denotes no limit.
spring.jta.atomikos.connectionfactory.unique-resource-name=jmsConnectionFactory # The unique name used to identify the resource during recovery.
spring.jta.atomikos.connectionfactory.xa-connection-factory-class-name= # Vendor-specific implementation of XAConnectionFactory.
spring.jta.atomikos.connectionfactory.xa-properties= # Vendor-specific XA properties.

```

spring.jta.atomikos.datasource.borrow-connection-timeout=30 # Timeout, in seconds, for borrowing connections from the pool.  
 spring.jta.atomikos.datasource.concurrent-connection-validation= # Whether to use concurrent connection validation.  
 spring.jta.atomikos.datasource.default-isolation-level= # Default isolation level of connections provided by the pool.  
 spring.jta.atomikos.datasource.login-timeout= # Timeout, in seconds, for establishing a database connection.  
 spring.jta.atomikos.datasource.maintenance-interval=60 # The time, in seconds, between runs of the pool's maintenance thread.  
 spring.jta.atomikos.datasource.max-idle-time=60 # The time, in seconds, after which connections are cleaned up from the pool.  
 spring.jta.atomikos.datasource.max-lifetime=0 # The time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit.  
 spring.jta.atomikos.datasource.max-pool-size=1 # The maximum size of the pool.  
 spring.jta.atomikos.datasource.min-pool-size=1 # The minimum size of the pool.  
 spring.jta.atomikos.datasource.reap-timeout=0 # The reap timeout, in seconds, for borrowed connections. 0 denotes no limit.  
 spring.jta.atomikos.datasource.test-query= # SQL query or statement used to validate a connection before returning it.  
 spring.jta.atomikos.datasource.unique-resource-name=dataSource # The unique name used to identify the resource during recovery.  
 spring.jta.atomikos.datasource.xa-data-source-class-name= # Vendor-specific implementation of XAConnectionFactory.  
 spring.jta.atomikos.datasource.xa-properties= # Vendor-specific XA properties.  
 spring.jta.atomikos.properties.allow-sub-transactions=true # Specify whether sub-transactions are allowed.  
 spring.jta.atomikos.properties.checkpoint-interval=500 # Interval between checkpoints, expressed as the number of log writes between two checkpoints.  
 spring.jta.atomikos.properties.default-jta-timeout=10000ms # Default timeout for JTA transactions.  
 spring.jta.atomikos.properties.default-max-wait-time-on-shutdown=9223372036854775807 # How long should normal shutdown (no-force) wait for transactions to complete.  
 spring.jta.atomikos.properties.enable-logging=true # Whether to enable disk logging.  
 spring.jta.atomikos.properties.force-shutdown-on-vm-exit=false # Whether a VM shutdown should trigger forced shutdown of the transaction core.  
 spring.jta.atomikos.properties.log-base-dir= # Directory in which the log files should be stored.  
 spring.jta.atomikos.properties.log-base-name=tmlog # Transactions log file base name.  
 spring.jta.atomikos.properties.max-actives=50 # Maximum number of active transactions.  
 spring.jta.atomikos.properties.max-timeout=300000ms # Maximum timeout that can be allowed for transactions.  
 spring.jta.atomikos.properties.recovery.delay=10000ms # Delay between two recovery scans.  
 spring.jta.atomikos.properties.recovery.forget-orphaned-log-entries-delay=86400000ms # Delay after which recovery can cleanup pending ('orphaned') log entries.  
 spring.jta.atomikos.properties.recovery.max-retries=5 # Number of retry attempts to commit the transaction before throwing an exception.  
 spring.jta.atomikos.properties.recovery.retry-interval=10000ms # Delay between retry attempts.  
 spring.jta.atomikos.properties.serial-jta-transactions=true # Whether sub-transactions should be joined when possible.  
 spring.jta.atomikos.properties.service= # Transaction manager implementation that should be started.  
 spring.jta.atomikos.properties.threaded-two-phase-commit=false # Whether to use different (and concurrent) threads for two-phase commit on the participating resources.  
 spring.jta.atomikos.properties.transaction-manager-unique-name= # The transaction manager's unique name.

**# BITRONIX**  
 spring.jta.bitronix.connectionfactory.acquire-increment=1 # Number of connections to create when growing the pool.  
 spring.jta.bitronix.connectionfactory.acquisition-interval=1 # Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired.  
 spring.jta.bitronix.connectionfactory.acquisition-timeout=30 # Timeout, in seconds, for acquiring connections from the pool.  
 spring.jta.bitronix.connectionfactory.allow-local-transactions=true # Whether the transaction manager should allow mixing XA and non-XA transactions.  
 spring.jta.bitronix.connectionfactory.apply-transaction-timeout=false # Whether the transaction timeout should be set on the XAResource when it is enlisted.  
 spring.jta.bitronix.connectionfactory.automatic-enlisting-enabled=true # Whether resources should be enlisted and delisted automatically.  
 spring.jta.bitronix.connectionfactory.cache-producers-consumers=true # Whether producers and consumers should be cached.  
 spring.jta.bitronix.connectionfactory.class-name= # Underlying implementation class name of the XA resource.  
 spring.jta.bitronix.connectionfactory.defer-connection-release=true # Whether the provider can run many transactions on the same connection and supports transaction interleaving.  
 spring.jta.bitronix.connectionfactory.disabled= # Whether this resource is disabled, meaning it's temporarily forbidden to acquire a connection from its pool.  
 spring.jta.bitronix.connectionfactory.driver-properties= # Properties that should be set on the underlying implementation.  
 spring.jta.bitronix.connectionfactory.failed= # Mark this resource producer as failed.  
 spring.jta.bitronix.connectionfactory.ignore-recovery-failures=false # Whether recovery failures should be ignored.  
 spring.jta.bitronix.connectionfactory.max-idle-time=60 # The time, in seconds, after which connections are cleaned up from the pool.  
 spring.jta.bitronix.connectionfactory.max-pool-size=10 # The maximum size of the pool. 0 denotes no limit.  
 spring.jta.bitronix.connectionfactory.min-pool-size=0 # The minimum size of the pool.  
 spring.jta.bitronix.connectionfactory.password= # The password to use to connect to the JMS provider.  
 spring.jta.bitronix.connectionfactory.share-transaction-connections=false # Whether connections in the ACCESSIBLE state can be shared within the context of a transaction.  
 spring.jta.bitronix.connectionfactory.test-connections=true # Whether connections should be tested when acquired from the pool.  
 spring.jta.bitronix.connectionfactory.two-pc-ordering-position=1 # The position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, always last is Integer.MAX\_VALUE).  
 spring.jta.bitronix.connectionfactory.unique-name=jmsConnectionFactory # The unique name used to identify the resource during recovery.  
 spring.jta.bitronix.connectionfactory.use-tm-join=true # Whether TMJOIN should be used when starting XAResources.  
 spring.jta.bitronix.connectionfactory.user= # The user to use to connect to the JMS provider.  
 spring.jta.bitronix.datasource.acquire-increment=1 # Number of connections to create when growing the pool.  
 spring.jta.bitronix.datasource.acquisition-interval=1 # Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired.  
 spring.jta.bitronix.datasource.acquisition-timeout=30 # Timeout, in seconds, for acquiring connections from the pool.  
 spring.jta.bitronix.datasource.allow-local-transactions=true # Whether the transaction manager should allow mixing XA and non-XA transactions.  
 spring.jta.bitronix.datasource.apply-transaction-timeout=false # Whether the transaction timeout should be set on the XAResource when it is enlisted.  
 spring.jta.bitronix.datasource.automatic-enlisting-enabled=true # Whether resources should be enlisted and delisted automatically.  
 spring.jta.bitronix.datasource.class-name= # Underlying implementation class name of the XA resource.  
 spring.jta.bitronix.datasource.cursor-holdability= # The default cursor holdability for connections.  
 spring.jta.bitronix.datasource.defer-connection-release=true # Whether the database can run many transactions on the same connection and supports transaction interleaving.  
 spring.jta.bitronix.datasource.disabled= # Whether this resource is disabled, meaning it's temporarily forbidden to acquire a connection from its pool.  
 spring.jta.bitronix.datasource.driver-properties= # Properties that should be set on the underlying implementation.  
 spring.jta.bitronix.datasource.enable-jdbc4-connection-test= # Whether Connection.isValid() is called when acquiring a connection from the pool.  
 spring.jta.bitronix.datasource.failed= # Mark this resource producer as failed.  
 spring.jta.bitronix.datasource.ignore-recovery-failures=false # Whether recovery failures should be ignored.  
 spring.jta.bitronix.datasource.isolation-level= # The default isolation level for connections.  
 spring.jta.bitronix.datasource.local-auto-commit= # The default auto-commit mode for local transactions.  
 spring.jta.bitronix.datasource.login-timeout= # Timeout, in seconds, for establishing a database connection.  
 spring.jta.bitronix.datasource.max-idle-time=60 # The time, in seconds, after which connections are cleaned up from the pool.  
 spring.jta.bitronix.datasource.max-pool-size=10 # The maximum size of the pool. 0 denotes no limit.  
 spring.jta.bitronix.datasource.min-pool-size=0 # The minimum size of the pool.  
 spring.jta.bitronix.datasource.prepared-statement-cache-size=0 # The target size of the prepared statement cache. 0 disables the cache.  
 spring.jta.bitronix.datasource.share-transaction-connections=false # Whether connections in the ACCESSIBLE state can be shared within the context of a transaction.  
 spring.jta.bitronix.datasource.test-query= # SQL query or statement used to validate a connection before returning it.  
 spring.jta.bitronix.datasource.two-pc-ordering-position=1 # The position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, and always last is Integer.MAX\_VALUE).  
 spring.jta.bitronix.datasource.unique-name=dataSource # The unique name used to identify the resource during recovery.  
 spring.jta.bitronix.datasource.use-tm-join=true # Whether TMJOIN should be used when starting XAResources.  
 spring.jta.bitronix.properties.allow-multiple-lrc=false # Whether to allow multiple LRC resources to be enlisted into the same transaction.  
 spring.jta.bitronix.properties.asynchronous2-pc=false # Whether to enable asynchronously execution of two phase commit.  
 spring.jta.bitronix.properties.background-recovery-interval-seconds=60 # Interval in seconds at which to run the recovery process in the background.

spring.jta.bitronix.properties.current-node-only-recovery=true # Whether to recover only the current node.  
spring.jta.bitronix.properties.debug-zero-resource-transaction=false # Whether to log the creation and commit call stacks of transactions executed without a single enlisted resource.  
spring.jta.bitronix.properties.default-transaction-timeout=60 # Default transaction timeout, in seconds.  
spring.jta.bitronix.properties.disable-jmx=false # Whether to enable JMX support.  
spring.jta.bitronix.properties.exception-analyzer= # Set the fully qualified name of the exception analyzer implementation to use.  
spring.jta.bitronix.properties.filter-log-status=false # Whether to enable filtering of logs so that only mandatory logs are written.  
spring.jta.bitronix.properties.force-batching-enabled=true # Whether disk forces are batched.  
spring.jta.bitronix.properties.forced-write-enabled=true # Whether logs are forced to disk.  
spring.jta.bitronix.properties.graceful-shutdown-interval=60 # Maximum amount of seconds the TM waits for transactions to get done before aborting them at shutdown time.  
spring.jta.bitronix.properties.jndi-transaction-synchronization-registry-name= # JNDI name of the TransactionSynchronizationRegistry.  
spring.jta.bitronix.properties.jndi-user-transaction-name= # JNDI name of the UserTransaction.  
spring.jta.bitronix.properties.journal=disk # Name of the journal. Can be 'disk', 'null', or a class name.  
spring.jta.bitronix.properties.log-part1-filename=btm1.tlog # Name of the first fragment of the journal.  
spring.jta.bitronix.properties.log-part2-filename=btm2.tlog # Name of the second fragment of the journal.  
spring.jta.bitronix.properties.max-log-size-in-mb=2 # Maximum size in megabytes of the journal fragments.  
spring.jta.bitronix.properties.resource-configuration-filename= # ResourceLoader configuration file name.  
spring.jta.bitronix.properties.server-id= # ASCII ID that must uniquely identify this TM instance. Defaults to the machine's IP address.  
spring.jta.bitronix.properties.skip-corrupted-logs=false # Skip corrupted transactions log entries.  
spring.jta.bitronix.properties.warn-about-zero-resource-transaction=true # Whether to log a warning for transactions executed without a single enlisted resource.

#### # NARAYANA (NarayanaProperties)

spring.jta.narayana.default-timeout=60s # Transaction timeout. If a duration suffix is not specified, seconds will be used.  
spring.jta.narayana.expiry-scanners=com.arjuna.ats.internal.arjuna.recovery.ExpiredTransactionStatusManagerScanner # Comma-separated list of expiry scanners.  
spring.jta.narayana.log-dir= # Transaction object store directory.  
spring.jta.narayana.one-phase-commit=true # Whether to enable one phase commit optimization.  
spring.jta.narayana.periodic-recovery-period=120s # Interval in which periodic recovery scans are performed. If a duration suffix is not specified, seconds will be used.  
spring.jta.narayana.recovery-backoff-period=10s # Back off period between first and second phases of the recovery scan. If a duration suffix is not specified, seconds will be used.  
spring.jta.narayana.recovery-db-pass= # Database password to be used by the recovery manager.  
spring.jta.narayana.recovery-db-user= # Database username to be used by the recovery manager.  
spring.jta.narayana.recovery-jms-pass= # JMS password to be used by the recovery manager.  
spring.jta.narayana.recovery-jms-user= # JMS username to be used by the recovery manager.  
spring.jta.narayana.recovery-modules= # Comma-separated list of recovery modules.  
spring.jta.narayana.transaction-manager-id=1 # Unique transaction manager id.  
spring.jta.narayana.xa-resource-orphan-filters= # Comma-separated list of orphan filters.

#### # EMBEDDED MONGODB (EmbeddedMongoProperties)

spring.mongodb.embedded.features=sync\_delay # Comma-separated list of features to enable.  
spring.mongodb.embedded.storage.database-dir= # Directory used for data storage.  
spring.mongodb.embedded.storage.oplog-size= # Maximum size of the oplog, in megabytes.  
spring.mongodb.embedded.storage.repl-set-name= # Name of the replica set.  
spring.mongodb.embedded.version=3.2.2 # Version of Mongo to use.

#### # REDIS (RedisProperties)

spring.redis.cluster.max-redirects= # Maximum number of redirects to follow when executing commands across the cluster.  
spring.redis.cluster.nodes= # Comma-separated list of "host:port" pairs to bootstrap from.  
spring.redis.database=0 # Database index used by the connection factory.  
spring.redis.url= # Connection URL. Overrides host, port, and password. User is ignored. Example: redis://user:password@example.com:6379  
spring.redis.host=localhost # Redis server host.  
spring.redis.jedis.pool.max-active=8 # Maximum number of connections that can be allocated by the pool at a given time. Use a negative value for no limit.  
spring.redis.jedis.pool.max-idle=8 # Maximum number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections.  
spring.redis.jedis.pool.max-wait=-1ms # Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely.  
spring.redis.jedis.pool.min-idle=0 # Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if it is positive.  
spring.redis.lettuce.pool.max-active=8 # Maximum number of connections that can be allocated by the pool at a given time. Use a negative value for no limit.  
spring.redis.lettuce.pool.max-idle=8 # Maximum number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections.  
spring.redis.lettuce.pool.max-wait=-1ms # Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely.  
spring.redis.lettuce.pool.min-idle=0 # Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if it is positive.  
spring.redis.lettuce.shutdown-timeout=100ms # Shutdown timeout.  
spring.redis.password= # Login password of the redis server.  
spring.redis.port=6379 # Redis server port.  
spring.redis.sentinel.master= # Name of the Redis server.  
spring.redis.sentinel.nodes= # Comma-separated list of "host:port" pairs.  
spring.redis.ssl=false # Whether to enable SSL support.  
spring.redis.timeout= # Connection timeout.

#### # TRANSACTION (TransactionProperties)

spring.transaction.default-timeout= # Default transaction timeout. If a duration suffix is not specified, seconds will be used.  
spring.transaction.rollback-on-commit-failure= # Whether to roll back on commit failures.



# Properties (Integration Properties)

18 December 2023 01:07 PM

```
# -----
# INTEGRATION PROPERTIES
# -----

# ACTIVEMQ (ActiveMQProperties)
spring.activemq.broker-url= # URL of the ActiveMQ broker. Auto-generated by default.
spring.activemq.close-timeout=15s # Time to wait before considering a close complete.
spring.activemq.in-memory=true # Whether the default broker URL should be in memory. Ignored if an explicit broker has been specified.
spring.activemq.non-blocking-redelivery=false # Whether to stop message delivery before re-delivering messages from a rolled back transaction. This implies that message order is not preserved when this is enabled.
spring.activemq.password= # Login password of the broker.
spring.activemq.send-timeout=0ms # Time to wait on message sends for a response. Set it to 0 to wait forever.
spring.activemq.user= # Login user of the broker.
spring.activemq.packages.trust-all= # Whether to trust all packages.
spring.activemq.packages.trusted= # Comma-separated list of specific packages to trust (when not trusting all packages).
spring.activemq.pool.block-if-full=true # Whether to block when a connection is requested and the pool is full. Set it to false to throw a "JMSEException" instead.
spring.activemq.pool.block-if-full-timeout=-1ms # Blocking period before throwing an exception if the pool is still full.
spring.activemq.pool.create-connection-on-startup=true # Whether to create a connection on startup. Can be used to warm up the pool on startup.
spring.activemq.pool.enabled=false # Whether a PooledConnectionFactory should be created, instead of a regular ConnectionFactory.
spring.activemq.pool.expiry-timeout=0ms # Connection expiration timeout.
spring.activemq.pool.idle-timeout=30s # Connection idle timeout.
spring.activemq.pool.max-connections=1 # Maximum number of pooled connections.
spring.activemq.pool.maximum-active-session-per-connection=500 # Maximum number of active sessions per connection.
spring.activemq.pool.reconnect-on-exception=true # Reset the connection when a "JMSEException" occurs.
spring.activemq.pool.time-between-expiration-check=-1ms # Time to sleep between runs of the idle connection eviction thread. When negative, no idle connection eviction thread runs.
spring.activemq.pool.use-anonymous-producers=true # Whether to use only one anonymous "MessageProducer" instance. Set it to false to create one "MessageProducer" every time one is required.

# ARTEMIS (ArtemisProperties)
spring.artemis.embedded.cluster-password= # Cluster password. Randomly generated on startup by default.
spring.artemis.embedded.data-directory= # Journal file directory. Not necessary if persistence is turned off.
spring.artemis.embedded.enabled=true # Whether to enable embedded mode if the Artemis server APIs are available.
spring.artemis.embedded.persistent=false # Whether to enable persistent store.
spring.artemis.embedded.queues= # Comma-separated list of queues to create on startup.
spring.artemis.embedded.server-id= # Server ID. By default, an auto-incremented counter is used.
spring.artemis.embedded.topics= # Comma-separated list of topics to create on startup.
spring.artemis.host=localhost # Artemis broker host.
spring.artemis.mode= # Artemis deployment mode, auto-detected by default.
spring.artemis.password= # Login password of the broker.
spring.artemis.port=61616 # Artemis broker port.
spring.artemis.user= # Login user of the broker.

# SPRING BATCH (BatchProperties)
spring.batch.initialize-schema=embedded # Database schema initialization mode.
spring.batch.job.enabled=true # Execute all Spring Batch jobs in the context on startup.
spring.batch.job.names= # Comma-separated list of job names to execute on startup (for instance, 'job1,job2'). By default, all Jobs found in the context are executed.
spring.batch.schema=classpath:org/springframework/batch/core/schema-@@platform@@.sql # Path to the SQL file to use to initialize the database schema.
spring.batch.table-prefix= # Table prefix for all the batch meta-data tables.

# SPRING INTEGRATION (IntegrationProperties)
spring.integration.jdbc.initialize-schema=embedded # Database schema initialization mode.
spring.integration.jdbc.schema=classpath:org/springframework/integration/jdbc/schema-@@platform@@.sql # Path to the SQL file to use to initialize the database schema.

# JMS (JmsProperties)
spring.jms.jndi-name= # Connection factory JNDI name. When set, takes precedence to others connection factory auto-configurations.
spring.jms.listener.acknowledge-mode= # Acknowledge mode of the container. By default, the listener is transacted with automatic acknowledgment.
spring.jms.listener.auto-startup=true # Start the container automatically on startup.
spring.jms.listener.concurrency= # Minimum number of concurrent consumers.
spring.jms.listener.max-concurrency= # Maximum number of concurrent consumers.
spring.jms.pub-sub-domain=false # Whether the default destination type is topic.
spring.jms.template.default-destination= # Default destination to use on send and receive operations that do not have a destination parameter.
spring.jms.template.delivery-delay= # Delivery delay to use for send calls.
spring.jms.template.delivery-mode= # Delivery mode. Enables QoS (Quality of Service) when set.
spring.jms.template.priority= # Priority of a message when sending. Enables QoS (Quality of Service) when set.
spring.jms.template.qos-enabled= # Whether to enable explicit QoS (Quality of Service) when sending a message.
spring.jms.template.receive-timeout= # Timeout to use for receive calls.
spring.jms.template.time-to-live= # Time-to-live of a message when sending. Enables QoS (Quality of Service) when set.

# APACHE KAFKA (KafkaProperties)
spring.kafka.admin.client-id= # ID to pass to the server when making requests. Used for server-side logging.
spring.kafka.admin.fail-fast=false # Whether to fail fast if the broker is not available on startup.
spring.kafka.admin.properties.*= # Additional admin-specific properties used to configure the client.
spring.kafka.admin.ssl.key-password= # Password of the private key in the key store file.
spring.kafka.admin.ssl.keystore-location= # Location of the key store file.
spring.kafka.admin.ssl.keystore-password= # Store password for the key store file.
spring.kafka.admin.ssl.keystore-type= # Type of the key store.
spring.kafka.admin.ssl.protocol= # SSL protocol to use.
spring.kafka.admin.ssl.truststore-location= # Location of the trust store file.
spring.kafka.admin.ssl.truststore-password= # Store password for the trust store file.
spring.kafka.admin.ssl.truststore-type= # Type of the trust store.
spring.kafka.bootstrap-servers= # Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.
spring.kafka.client-id= # ID to pass to the server when making requests. Used for server-side logging.
spring.kafka.consumer.auto-commit-interval= # Frequency with which the consumer offsets are auto-committed to Kafka if 'enable.auto.commit' is set to true.
spring.kafka.consumer.auto-offset-reset= # What to do when there is no initial offset in Kafka or if the current offset no longer exists on the server.
spring.kafka.consumer.bootstrap-servers= # Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.
spring.kafka.consumer.client-id= # ID to pass to the server when making requests. Used for server-side logging.
spring.kafka.consumer.enable-auto-commit= # Whether the consumer's offset is periodically committed in the background.
spring.kafka.consumer.fetch-max-wait= # Maximum amount of time the server blocks before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement
```

given by "fetch.min.bytes".

spring.kafka.consumer.fetch-min-size= # Minimum amount of data, in bytes, the server should return for a fetch request.

spring.kafka.consumer.group-id= # Unique string that identifies the consumer group to which this consumer belongs.

spring.kafka.consumer.heartbeat-interval= # Expected time between heartbeats to the consumer coordinator.

spring.kafka.consumer.key-deserializer= # Deserializer class for keys.

spring.kafka.consumer.max-poll-records= # Maximum number of records returned in a single call to poll().

spring.kafka.consumer.properties.\*= # Additional consumer-specific properties used to configure the client.

spring.kafka.consumer.ssl.key-password= # Password of the private key in the key store file.

spring.kafka.consumer.ssl.keystore-location= # Location of the key store file.

spring.kafka.consumer.ssl.keystore-password= # Store password for the key store file.

spring.kafka.consumer.ssl.keystore-type= # Type of the key store.

spring.kafka.consumer.ssl.protocol= # SSL protocol to use.

spring.kafka.consumer.ssl.truststore-location= # Location of the trust store file.

spring.kafka.consumer.ssl.truststore-password= # Store password for the trust store file.

spring.kafka.consumer.ssl.truststore-type= # Type of the trust store.

spring.kafka.consumer.value-deserializer= # Deserializer class for values.

spring.kafka.jaas.control-flag=required # Control flag for login configuration.

spring.kafka.jaas.enabled=false # Whether to enable JAAS configuration.

spring.kafka.jaas.login-module=com.sun.security.auth.module.Krb5LoginModule # Login module.

spring.kafka.jaas.options= # Additional JAAS options.

spring.kafka.listener.ack-count= # Number of records between offset commits when ackMode is "COUNT" or "COUNT\_TIME".

spring.kafka.listener.ack-mode= # Listener AckMode. See the spring-kafka documentation.

spring.kafka.listener.ack-time= # Time between offset commits when ackMode is "TIME" or "COUNT\_TIME".

spring.kafka.listener.client-id= # Prefix for the listener's consumer client.id property.

spring.kafka.listener.concurrency= # Number of threads to run in the listener containers.

spring.kafka.listener.idle-event-interval= # Time between publishing idle consumer events (no data received).

spring.kafka.listener.log-container-config= # Whether to log the container configuration during initialization (INFO level).

spring.kafka.listener.monitor-interval= # Time between checks for non-responsive consumers. If a duration suffix is not specified, seconds will be used.

spring.kafka.listener.no-poll-threshold= # Multiplier applied to "pollTimeout" to determine if a consumer is non-responsive.

spring.kafka.listener.poll-timeout= # Timeout to use when polling the consumer.

spring.kafka.listener.type=single # Listener type.

spring.kafka.producer.acks= # Number of acknowledgments the producer requires the leader to have received before considering a request complete.

spring.kafka.producer.batch-size= # Default batch size in bytes.

spring.kafka.producer.bootstrap-servers= # Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.

spring.kafka.producer.buffer-memory= # Total bytes of memory the producer can use to buffer records waiting to be sent to the server.

spring.kafka.producer.client-id= # ID to pass to the server when making requests. Used for server-side logging.

spring.kafka.producer.compression-type= # Compression type for all data generated by the producer.

spring.kafka.producer.key-serializer= # Serializer class for keys.

spring.kafka.producer.properties.\*= # Additional producer-specific properties used to configure the client.

spring.kafka.producer.retries= # When greater than zero, enables retrying of failed sends.

spring.kafka.producer.ssl.key-password= # Password of the private key in the key store file.

spring.kafka.producer.ssl.keystore-location= # Location of the key store file.

spring.kafka.producer.ssl.keystore-password= # Store password for the key store file.

spring.kafka.producer.ssl.keystore-type= # Type of the key store.

spring.kafka.producer.ssl.protocol= # SSL protocol to use.

spring.kafka.producer.ssl.truststore-location= # Location of the trust store file.

spring.kafka.producer.ssl.truststore-password= # Store password for the trust store file.

spring.kafka.producer.ssl.truststore-type= # Type of the trust store.

spring.kafka.producer.transaction-id-prefix= # When non empty, enables transaction support for producer.

spring.kafka.producer.value-serializer= # Serializer class for values.

spring.kafka.properties.\*= # Additional properties, common to producers and consumers, used to configure the client.

spring.kafka.ssl.key-password= # Password of the private key in the key store file.

spring.kafka.ssl.keystore-location= # Location of the key store file.

spring.kafka.ssl.keystore-password= # Store password for the key store file.

spring.kafka.ssl.keystore-type= # Type of the key store.

spring.kafka.ssl.protocol= # SSL protocol to use.

spring.kafka.ssl.truststore-location= # Location of the trust store file.

spring.kafka.ssl.truststore-password= # Store password for the trust store file.

spring.kafka.ssl.truststore-type= # Type of the trust store.

spring.kafka.template.default-topic= # Default topic to which messages are sent.

#### # RABBIT (RabbitProperties)

spring.rabbitmq.addresses= # Comma-separated list of addresses to which the client should connect.

spring.rabbitmq.cache.channel.checkout-timeout= # Duration to wait to obtain a channel if the cache size has been reached.

spring.rabbitmq.cache.channel.size= # Number of channels to retain in the cache.

spring.rabbitmq.cache.connection.mode=channel # Connection factory cache mode.

spring.rabbitmq.cache.connection.size= # Number of connections to cache.

spring.rabbitmq.connection-timeout= # Connection timeout. Set it to zero to wait forever.

spring.rabbitmq.dynamic=true # Whether to create an AmqpAdmin bean.

spring.rabbitmq.host=localhost # RabbitMQ host.

spring.rabbitmq.listener.direct.acknowledge-mode= # Acknowledge mode of container.

spring.rabbitmq.listener.direct.auto-startup=true # Whether to start the container automatically on startup.

spring.rabbitmq.listener.direct.consumers-per-queue= # Number of consumers per queue.

spring.rabbitmq.listener.direct.default-requeue-rejected= # Whether rejected deliveries are re-queued by default.

spring.rabbitmq.listener.direct.idle-event-interval= # How often idle container events should be published.

spring.rabbitmq.listener.direct.prefetch= # Number of messages to be handled in a single request. It should be greater than or equal to the transaction size (if used).

spring.rabbitmq.listener.direct.retry.enabled=false # Whether publishing retries are enabled.

spring.rabbitmq.listener.direct.retry.initial-interval=1000ms # Duration between the first and second attempt to deliver a message.

spring.rabbitmq.listener.direct.retry.max-attempts=3 # Maximum number of attempts to deliver a message.

spring.rabbitmq.listener.direct.retry.max-interval=10000ms # Maximum duration between attempts.

spring.rabbitmq.listener.direct.retry.multiplier=1 # Multiplier to apply to the previous retry interval.

spring.rabbitmq.listener.direct.retry.stateless=true # Whether retries are stateless or stateful.

spring.rabbitmq.listener.simple.acknowledge-mode= # Acknowledge mode of container.

spring.rabbitmq.listener.simple.auto-startup=true # Whether to start the container automatically on startup.

spring.rabbitmq.listener.simple.concurrency= # Minimum number of listener invoker threads.

spring.rabbitmq.listener.simple.default-requeue-rejected= # Whether rejected deliveries are re-queued by default.

spring.rabbitmq.listener.simple.idle-event-interval= # How often idle container events should be published.

spring.rabbitmq.listener.simple.max-concurrency= # Maximum number of listener invoker threads.

spring.rabbitmq.listener.simple.prefetch= # Number of messages to be handled in a single request. It should be greater than or equal to the transaction size (if used).

spring.rabbitmq.listener.simple.retry.enabled=false # Whether publishing retries are enabled.

spring.rabbitmq.listener.simple.retry.initial-interval=1000ms # Duration between the first and second attempt to deliver a message.

spring.rabbitmq.listener.simple.retry.max-attempts=3 # Maximum number of attempts to deliver a message.

spring.rabbitmq.listener.simple.retry.max-interval=10000ms # Maximum duration between attempts.

spring.rabbitmq.listener.simple.retry.multiplier=1 # Multiplier to apply to the previous retry interval.

spring.rabbitmq.listener.simple.retry.stateless=true # Whether retries are stateless or stateful.

spring.rabbitmq.listener.simple.transaction-size= # Number of messages to be processed in a transaction. That is, the number of messages between acks. For best results, it should be less than or equal to the prefetch count.

spring.rabbitmq.listener.type=simple # Listener container type.

spring.rabbitmq.password=guest # Login to authenticate against the broker.

spring.rabbitmq.port=5672 # RabbitMQ port.

spring.rabbitmq.publisher-confirms=false # Whether to enable publisher confirms.

spring.rabbitmq.publisher-returns=false # Whether to enable publisher returns.

spring.rabbitmq.requested-heartbeat= # Requested heartbeat timeout; zero for none. If a duration suffix is not specified, seconds will be used.

spring.rabbitmq.ssl.enabled=false # Whether to enable SSL support.

spring.rabbitmq.ssl.key-store= # Path to the key store that holds the SSL certificate.

spring.rabbitmq.ssl.key-store-password= # Password used to access the key store.

spring.rabbitmq.ssl.key-store-type=PKCS12 # Key store type.

spring.rabbitmq.ssl.trust-store= # Trust store that holds SSL certificates.

spring.rabbitmq.ssl.trust-store-password= # Password used to access the trust store.

spring.rabbitmq.ssl.trust-store-type=JKS # Trust store type.

spring.rabbitmq.ssl.algorithm= # SSL algorithm to use. By default, configured by the Rabbit client library.

spring.rabbitmq.template.exchange= # Name of the default exchange to use for send operations.

spring.rabbitmq.template.mandatory= # Whether to enable mandatory messages.

spring.rabbitmq.template.receive-timeout= # Timeout for 'receive()' operations.

spring.rabbitmq.template.reply-timeout= # Timeout for 'sendAndReceive()' operations.

spring.rabbitmq.template.retry.enabled=false # Whether publishing retries are enabled.

spring.rabbitmq.template.retry.initial-interval=1000ms # Duration between the first and second attempt to deliver a message.

spring.rabbitmq.template.retry.max-attempts=3 # Maximum number of attempts to deliver a message.

spring.rabbitmq.template.retry.max-interval=10000ms # Maximum duration between attempts.

spring.rabbitmq.template.retry.multiplier=1 # Multiplier to apply to the previous retry interval.

spring.rabbitmq.template.routing-key= # Value of a default routing key to use for send operations.

spring.rabbitmq.username=guest # Login user to authenticate to the broker.

spring.rabbitmq.virtual-host= # Virtual host to use when connecting to the broker.

# Properties(Actuator Properties)

18 December 2023 01:08 PM

```
# -----
# ACTUATOR PROPERTIES
# -----

# MANAGEMENT HTTP SERVER (ManagementServerProperties)
management.server.add-application-context-header=false # Add the "X-Application-Context" HTTP header in each response.
management.server.address= # Network address to which the management endpoints should bind. Requires a custom management.server.port.
management.server.port= # Management endpoint HTTP port (uses the same port as the application by default). Configure a different port to use management-specific SSL.
management.server.servlet.context-path= # Management endpoint context-path (for instance, '/management'). Requires a custom management.server.port.
management.server.ssl.ciphers= # Supported SSL ciphers. Requires a custom management.server.port.
management.server.ssl.client-auth= # Whether client authentication is wanted ("want") or needed ("need"). Requires a trust store. Requires a custom management.server.port.
management.server.ssl.enabled= # Whether to enable SSL support. Requires a custom management.server.port.
management.server.ssl.enabled-protocols= # Enabled SSL protocols. Requires a custom management.server.port.
management.server.ssl.key-alias= # Alias that identifies the key in the key store. Requires a custom management.server.port.
management.server.ssl.key-password= # Password used to access the key in the key store. Requires a custom management.server.port.
management.server.ssl.key-store= # Path to the key store that holds the SSL certificate (typically a jks file). Requires a custom management.server.port.
management.server.ssl.key-store-password= # Password used to access the key store. Requires a custom management.server.port.
management.server.ssl.key-store-provider= # Provider for the key store. Requires a custom management.server.port.
management.server.ssl.key-store-type= # Type of the key store. Requires a custom management.server.port.
management.server.ssl.protocol=TLS # SSL protocol to use. Requires a custom management.server.port.
management.server.ssl.trust-store= # Trust store that holds SSL certificates. Requires a custom management.server.port.
management.server.ssl.trust-store-password= # Password used to access the trust store. Requires a custom management.server.port.
management.server.ssl.trust-store-provider= # Provider for the trust store. Requires a custom management.server.port.
management.server.ssl.trust-store-type= # Type of the trust store. Requires a custom management.server.port.

# CLOUDFOUNDRY
management.cloudfoundry.enabled=true # Whether to enable extended Cloud Foundry actuator endpoints.
management.cloudfoundry.skip-ssl-validation=false # Whether to skip SSL verification for Cloud Foundry actuator endpoint security calls.

# ENDPOINTS GENERAL CONFIGURATION
management.endpoints.enabled-by-default= # Whether to enable or disable all endpoints by default.

# ENDPOINTS JMX CONFIGURATION (JmxEndpointProperties)
management.endpoints.jmx.domain=org.springframework.boot # Endpoints JMX domain name. Fallback to 'spring.jmx.default-domain' if set.
management.endpoints.jmx.exposure.include=* # Endpoint IDs that should be included or '*' for all.
management.endpoints.jmx.exposure.exclude= # Endpoint IDs that should be excluded or '*' for all.
management.endpoints.jmx.static-names= # Additional static properties to append to all ObjectNames of MBeans representing Endpoints.
management.endpoints.jmx.unique-names=false # Whether to ensure that ObjectNames are modified in case of conflict.

# ENDPOINTS WEB CONFIGURATION (WebEndpointProperties)
management.endpoints.web.exposure.include=health,info # Endpoint IDs that should be included or '*' for all.
management.endpoints.web.exposure.exclude= # Endpoint IDs that should be excluded or '*' for all.
management.endpoints.web.base-path=/actuator # Base path for Web endpoints. Relative to server.servlet.context-path or management.server.servlet.context-path if management.server.port is configured.
management.endpoints.web.path-mapping= # Mapping between endpoint IDs and the path that should expose them.

# ENDPOINTS CORS CONFIGURATION (CorsEndpointProperties)
management.endpoints.web.cors.allow-credentials= # Whether credentials are supported. When not set, credentials are not supported.
management.endpoints.web.cors.allowed-headers= # Comma-separated list of headers to allow in a request. '*' allows all headers.
management.endpoints.web.cors.allowed-methods= # Comma-separated list of methods to allow. '*' allows all methods. When not set, defaults to GET.
management.endpoints.web.cors.allowed-origins= # Comma-separated list of origins to allow. '*' allows all origins. When not set, CORS support is disabled.
management.endpoints.web.cors.exposed-headers= # Comma-separated list of headers to include in a response.
management.endpoints.web.cors.max-age=1800s # How long the response from a pre-flight request can be cached by clients. If a duration suffix is not specified, seconds will be used.

# AUDIT EVENTS ENDPOINT (AuditEventsEndpoint)
management.endpoint.auditcache.time-to-live=0ms # Maximum time that a response can be cached.
management.endpoint.auditcache.enabled=true # Whether to enable the auditcache endpoint.

# BEANS ENDPOINT (BeansEndpoint)
management.endpoint.beans.cache.time-to-live=0ms # Maximum time that a response can be cached.
management.endpoint.beans.enabled=true # Whether to enable the beans endpoint.

# CONDITIONS REPORT ENDPOINT (ConditionsReportEndpoint)
management.endpoint.conditions.cache.time-to-live=0ms # Maximum time that a response can be cached.
management.endpoint.conditions.enabled=true # Whether to enable the conditions endpoint.

# CONFIGURATION PROPERTIES REPORT ENDPOINT (ConfigurationPropertiesReportEndpoint, ConfigurationPropertiesReportEndpointProperties)
management.endpoint.configprops.cache.time-to-live=0ms # Maximum time that a response can be cached.
management.endpoint.configprops.enabled=true # Whether to enable the configprops endpoint.
management.endpoint.configprops.keys-to-sanitize=password,secret,key,token,*credentials.*,*vcap_services,sun.java.command # Keys that should be sanitized. Keys can be simple strings that the property ends with or regular expressions.

# ENVIRONMENT ENDPOINT (EnvironmentEndpoint, EnvironmentEndpointProperties)
management.endpoint.env.cache.time-to-live=0ms # Maximum time that a response can be cached.
management.endpoint.env.enabled=true # Whether to enable the env endpoint.
```

management.endpoint.env.keys-to-sanitize=password,secret,key,token,.\*credentials.\*,vcap\_services,sun.java.command # Keys that should be sanitized. Keys can be simple strings that the property ends with or regular expressions.

#### # FLYWAY ENDPOINT (FlywayEndpoint)

management.endpoint.flyway.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.flyway.enabled=true # Whether to enable the flyway endpoint.

#### # HEALTH ENDPOINT (HealthEndpoint, HealthEndpointProperties)

management.endpoint.health.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.health.enabled=true # Whether to enable the health endpoint.  
management.endpoint.health.roles= # Roles used to determine whether or not a user is authorized to be shown details. When empty, all authenticated users are authorized.  
management.endpoint.health.show-details=never # When to show full health details.

#### # HEAP DUMP ENDPOINT (HeapDumpWebEndpoint)

management.endpoint.heapdump.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.heapdump.enabled=true # Whether to enable the heapdump endpoint.

#### # HTTP TRACE ENDPOINT (HttpTraceEndpoint)

management.endpoint.httptrace.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.httptrace.enabled=true # Whether to enable the httptrace endpoint.

#### # INFO ENDPOINT (InfoEndpoint)

info= # Arbitrary properties to add to the info endpoint.  
management.endpoint.info.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.info.enabled=true # Whether to enable the info endpoint.

#### # JOLOKIA ENDPOINT (JolokiaProperties)

management.endpoint.jolokia.config.\*= # Jolokia settings. Refer to the documentation of Jolokia for more details.  
management.endpoint.jolokia.enabled=true # Whether to enable the jolokia endpoint.

#### # LIQUIBASE ENDPOINT (LiquibaseEndpoint)

management.endpoint.liquibase.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.liquibase.enabled=true # Whether to enable the liquibase endpoint.

#### # LOG FILE ENDPOINT (LogFileWebEndpoint, LogFileWebEndpointProperties)

management.endpoint.logfile.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.logfile.enabled=true # Whether to enable the logfile endpoint.  
management.endpoint.logfile.external-file= # External Logfile to be accessed. Can be used if the logfile is written by output redirect and not by the logging system itself.

#### # LOGGERS ENDPOINT (LoggersEndpoint)

management.endpoint.loggers.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.loggers.enabled=true # Whether to enable the loggers endpoint.

#### # REQUEST MAPPING ENDPOINT (MappingsEndpoint)

management.endpoint.mappings.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.mappings.enabled=true # Whether to enable the mappings endpoint.

#### # METRICS ENDPOINT (MetricsEndpoint)

management.endpoint.metrics.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.metrics.enabled=true # Whether to enable the metrics endpoint.

#### # PROMETHEUS ENDPOINT (PrometheusScrapeEndpoint)

management.endpoint.prometheus.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.prometheus.enabled=true # Whether to enable the prometheus endpoint.

#### # SCHEDULED TASKS ENDPOINT (ScheduledTasksEndpoint)

management.endpoint.scheduledtasks.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.scheduledtasks.enabled=true # Whether to enable the scheduledtasks endpoint.

#### # SESSIONS ENDPOINT (SessionsEndpoint)

management.endpoint.sessions.enabled=true # Whether to enable the sessions endpoint.

#### # SHUTDOWN ENDPOINT (ShutdownEndpoint)

management.endpoint.shutdown.enabled=false # Whether to enable the shutdown endpoint.

#### # THREAD DUMP ENDPOINT (ThreadDumpEndpoint)

management.endpoint.threaddump.cache.time-to-live=0ms # Maximum time that a response can be cached.  
management.endpoint.threaddump.enabled=true # Whether to enable the threaddump endpoint.

#### # HEALTH INDICATORS

management.health.db.enabled=true # Whether to enable database health check.  
management.health.cassandra.enabled=true # Whether to enable Cassandra health check.  
management.health.couchbase.enabled=true # Whether to enable Couchbase health check.  
management.health.defaults.enabled=true # Whether to enable default health indicators.  
management.health.diskpace.enabled=true # Whether to enable disk space health check.  
management.health.diskpace.path= # Path used to compute the available disk space.  
management.health.diskpace.threshold=0 # Minimum disk space, in bytes, that should be available.  
management.health.elasticsearch.enabled=true # Whether to enable Elasticsearch health check.  
management.health.elasticsearch.indices= # Comma-separated index names.  
management.health.elasticsearch.response-timeout=100ms # Time to wait for a response from the cluster.  
management.health.influxdb.enabled=true # Whether to enable InfluxDB health check.  
management.health.jms.enabled=true # Whether to enable JMS health check.  
management.health ldap.enabled=true # Whether to enable LDAP health check.

management.health.mail.enabled=true # Whether to enable Mail health check.  
management.health.mongo.enabled=true # Whether to enable MongoDB health check.  
management.health.neo4j.enabled=true # Whether to enable Neo4j health check.  
management.health.rabbit.enabled=true # Whether to enable RabbitMQ health check.  
management.health.redis.enabled=true # Whether to enable Redis health check.  
management.health.solr.enabled=true # Whether to enable Solr health check.  
management.health.status.http-mapping= # Mapping of health statuses to HTTP status codes. By default, registered health statuses map to sensible defaults (for example, UP maps to 200).  
management.health.status.order=DOWN,OUT\_OF\_SERVICE,UP,UNKNOWN # Comma-separated list of health statuses in order of severity.

#### # HTTP TRACING (HttpTraceProperties)

management.trace.http.enabled=true # Whether to enable HTTP request-response tracing.  
management.trace.http.include=request-headers,response-headers,cookies,errors # Items to be included in the trace.

#### # INFO CONTRIBUTORS (InfoContributorProperties)

management.info.build.enabled=true # Whether to enable build info.  
management.info.defaults.enabled=true # Whether to enable default info contributors.  
management.info.env.enabled=true # Whether to enable environment info.  
management.info.git.enabled=true # Whether to enable git info.  
management.info.git.mode=simple # Mode to use to expose git information.

#### # METRICS

management.metrics.binders.files.enabled=true # Whether to enable files metrics.  
management.metrics.binders.integration.enabled=true # Whether to enable Spring Integration metrics.  
management.metrics.binders.jvm.enabled=true # Whether to enable JVM metrics.  
management.metrics.binders.logback.enabled=true # Whether to enable Logback metrics.  
management.metrics.binders.processor.enabled=true # Whether to enable processor metrics.  
management.metrics.binders.uptime.enabled=true # Whether to enable uptime metrics.  
management.metrics.distribution.percentiles-histogram.\*= # Whether meter IDs starting-with the specified name should be publish percentile histograms.  
management.metrics.distribution.percentiles.\*= # Specific computed non-aggregable percentiles to ship to the backend for meter IDs starting-with the specified name.  
management.metrics.distribution.sla.\*= # Specific SLA boundaries for meter IDs starting-with the specified name. The longest match wins, the key `all` can also be used to configure all meters.  
management.metrics.enable.\*= # Whether meter IDs starting-with the specified name should be enabled. The longest match wins, the key `all` can also be used to configure all meters.  
management.metrics.export.atlas.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.  
management.metrics.export.atlas.config-refresh-frequency=10s # Frequency for refreshing config settings from the LWC service.  
management.metrics.export.atlas.config-time-to-live=150s # Time to live for subscriptions from the LWC service.  
management.metrics.export.atlas.config-uri=http://localhost:7101/lwc/api/v1/expressions/local-dev # URI for the Atlas LWC endpoint to retrieve current subscriptions.  
management.metrics.export.atlas.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.atlas.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.atlas.eval-uri=http://localhost:7101/lwc/api/v1/evaluate # URI for the Atlas LWC endpoint to evaluate the data for a subscription.  
management.metrics.export.atlas.lwc-enabled=false # Whether to enable streaming to Atlas LWC.  
management.metrics.export.atlas.meter-time-to-live=15m # Time to live for meters that do not have any activity. After this period the meter will be considered expired and will not get reported.  
management.metrics.export.atlas.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.atlas.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.atlas.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.atlas.uri=http://localhost:7101/api/v1/publish # URI of the Atlas server.  
management.metrics.export.datadog.api-key= # Datadog API key.  
management.metrics.export.datadog.application-key= # Datadog application key. Not strictly required, but improves the Datadog experience by sending meter descriptions, types, and base units to Datadog.  
management.metrics.export.datadog.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.  
management.metrics.export.datadog.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.datadog.descriptions=true # Whether to publish descriptions metadata to Datadog. Turn this off to minimize the amount of metadata sent.  
management.metrics.export.datadog.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.datadog.host-tag=instance # Tag that will be mapped to "host" when shipping metrics to Datadog.  
management.metrics.export.datadog.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.datadog.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.datadog.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.datadog.uri=https://app.datadoghq.com # URI to ship metrics to. If you need to publish metrics to an internal proxy en-route to Datadog, you can define the location of the proxy with this.  
management.metrics.export.ganglia.addressing-mode=multicast # UDP addressing mode, either unicast or multicast.  
management.metrics.export.ganglia.duration-units=milliseconds # Base time unit used to report durations.  
management.metrics.export.ganglia.enabled=true # Whether exporting of metrics to Ganglia is enabled.  
management.metrics.export.ganglia.host=localhost # Host of the Ganglia server to receive exported metrics.  
management.metrics.export.ganglia.port=8649 # Port of the Ganglia server to receive exported metrics.  
management.metrics.export.ganglia.protocol-version=3.1 # Ganglia protocol version. Must be either 3.1 or 3.0.  
management.metrics.export.ganglia.rate-units=seconds # Base time unit used to report rates.  
management.metrics.export.ganglia.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.ganglia.time-to-live=1 # Time to live for metrics on Ganglia. Set the multi-cast Time-To-Live to be one greater than the number of hops (routers) between the hosts.  
management.metrics.export.graphite.duration-units=milliseconds # Base time unit used to report durations.  
management.metrics.export.graphite.enabled=true # Whether exporting of metrics to Graphite is enabled.  
management.metrics.export.graphite.host=localhost # Host of the Graphite server to receive exported metrics.  
management.metrics.export.graphite.port=2004 # Port of the Graphite server to receive exported metrics.  
management.metrics.export.graphite.protocol=pickled # Protocol to use while shipping data to Graphite.  
management.metrics.export.graphite.rate-units=seconds # Base time unit used to report rates.  
management.metrics.export.graphite.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.graphite.tags-as-prefix= # For the default naming convention, turn the specified tag keys into part of the metric prefix.  
management.metrics.export.influx.auto-create-db=true # Whether to create the Influx database if it does not exist before attempting to publish metrics to it.  
management.metrics.export.influx.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.

management.metrics.export.influx.compressed=true # Whether to enable GZIP compression of metrics batches published to Influx.  
management.metrics.export.influx.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.influx.consistency=one # Write consistency for each point.  
management.metrics.export.influx.db=mydb # Tag that will be mapped to "host" when shipping metrics to Influx.  
management.metrics.export.influx.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.influx.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.influx.password= # Login password of the Influx server.  
management.metrics.export.influx.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.influx.retention-duration= # Time period for which Influx should retain data in the current database.  
management.metrics.export.influx.retention-shard-duration= # Time range covered by a shard group.  
management.metrics.export.influx.retention-policy= # Retention policy to use (Influx writes to the DEFAULT retention policy if one is not specified).  
management.metrics.export.influx.retention-replication-factor= # How many copies of the data are stored in the cluster.  
management.metrics.export.influx.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.influx.uri=http://localhost:8086 # URI of the Influx server.  
management.metrics.export.influx.user-name= # Login user of the Influx server.  
management.metrics.export.jmx.domain=metrics # Metrics JMX domain name.  
management.metrics.export.jmx.enabled=true # Whether exporting of metrics to JMX is enabled.  
management.metrics.export.jmx.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.newrelic.account-id= # New Relic account ID.  
management.metrics.export.newrelic.api-key= # New Relic API key.  
management.metrics.export.newrelic.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.  
management.metrics.export.newrelic.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.newrelic.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.newrelic.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.newrelic.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.newrelic.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.newrelic.uri=https://insights-collector.newrelic.com # URI to ship metrics to.  
management.metrics.export.prometheus.descriptions=true # Whether to enable publishing descriptions as part of the scrape payload to Prometheus. Turn this off to minimize the amount of data sent on each scrape.  
management.metrics.export.prometheus.enabled=true # Whether exporting of metrics to Prometheus is enabled.  
management.metrics.export.prometheus.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.signalfx.access-token= # SignalFX access token.  
management.metrics.export.signalfx.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.  
management.metrics.export.signalfx.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.signalfx.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.signalfx.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.signalfx.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.signalfx.source= # Uniquely identifies the app instance that is publishing metrics to SignalFx. Defaults to the local host name.  
management.metrics.export.signalfx.step=10s # Step size (i.e. reporting frequency) to use.  
management.metrics.export.signalfx.uri=https://ingest.signalfx.com # URI to ship metrics to.  
management.metrics.export.simple.enabled=true # Whether, in the absence of any other exporter, exporting of metrics to an in-memory backend is enabled.  
management.metrics.export.simple.mode=cumulative # Counting mode.  
management.metrics.export.simple.step=1m # Step size (i.e. reporting frequency) to use.  
management.metrics.export.statsd.enabled=true # Whether exporting of metrics to StatsD is enabled.  
management.metrics.export.statsd.flavor=datadog # StatsD line protocol to use.  
management.metrics.export.statsd.host=localhost # Host of the StatsD server to receive exported metrics.  
management.metrics.export.statsd.max-packet-length=1400 # Total length of a single payload should be kept within your network's MTU.  
management.metrics.export.statsd.polling-frequency=10s # How often gauges will be polled. When a gauge is polled, its value is recalculated and if the value has changed (or publishUnchangedMeters is true), it is sent to the StatsD server.  
management.metrics.export.statsd.port=8125 # Port of the StatsD server to receive exported metrics.  
management.metrics.export.statsd.publish-unchanged-meters=true # Whether to send unchanged meters to the StatsD server.  
management.metrics.export.wavefront.api-token= # API token used when publishing metrics directly to the Wavefront API host.  
management.metrics.export.wavefront.batch-size=10000 # Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made.  
management.metrics.export.wavefront.connect-timeout=1s # Connection timeout for requests to this backend.  
management.metrics.export.wavefront.enabled=true # Whether exporting of metrics to this backend is enabled.  
management.metrics.export.wavefront.global-prefix= # Global prefix to separate metrics originating from this app's white box instrumentation from those originating from other Wavefront integrations when viewed in the Wavefront UI.  
management.metrics.export.wavefront.num-threads=2 # Number of threads to use with the metrics publishing scheduler.  
management.metrics.export.wavefront.read-timeout=10s # Read timeout for requests to this backend.  
management.metrics.export.wavefront.source= # Unique identifier for the app instance that is the source of metrics being published to Wavefront. Defaults to the local host name.  
management.metrics.export.wavefront.step=10s # Step size (i.e. reporting frequency) to use.  
management.metrics.export.wavefront.uri=https://longboard.wavefront.com # URI to ship metrics to.  
management.metrics.use-global-registry=true # Whether auto-configured MeterRegistry implementations should be bound to the global static registry on Metrics.  
management.metrics.web.client.max-uri-tags=100 # Maximum number of unique URI tag values allowed. After the max number of tag values is reached, metrics with additional tag values are denied by filter.  
management.metrics.web.client.requests-metric-name=http.client.requests # Name of the metric for sent requests.  
management.metrics.web.server.auto-time-requests=true # Whether requests handled by Spring MVC or WebFlux should be automatically timed.  
management.metrics.web.server.requests-metric-name=http.server.requests # Name of the metric for received requests.

# Properties (Devtools Properties)

18 December 2023 01:08 PM

```
# -----  
# DEVTOOLS PROPERTIES  
# -----
```

## # DEVTOOLS (DevToolsProperties)

```
spring.devtools.livereload.enabled=true # Whether to enable a livereload.com-compatible server.  
spring.devtools.livereload.port=35729 # Server port.  
spring.devtools.restart.additional-exclude= # Additional patterns that should be excluded from triggering a full restart.  
spring.devtools.restart.additional-paths= # Additional paths to watch for changes.  
spring.devtools.restart.enabled=true # Whether to enable automatic restart.  
spring.devtools.restart.exclude=META-INF/maven/**,META-INF/resources/**,resources/**,static/**,public/**,templates/**,**/*Test.class,**/*Tests.class,git.properties,META-INF/build-info.properties # Patterns that should be excluded from triggering a full restart.  
spring.devtools.restart.log-condition-evaluation-delta=true # Whether to log the condition evaluation delta upon restart.  
spring.devtools.restart.poll-interval=1s # Amount of time to wait between polling for classpath changes.  
spring.devtools.restart.quiet-period=400ms # Amount of quiet time required without any classpath changes before a restart is triggered.  
spring.devtools.restart.trigger-file= # Name of a specific file that, when changed, triggers the restart check. If not specified, any classpath file change triggers the restart.
```

## # REMOTE DEVTOOLS (RemoteDevToolsProperties)

```
spring.devtools.remote.context-path=/.~spring-boot!~ # Context path used to handle the remote connection.  
spring.devtools.remote.proxy.host= # The host of the proxy to use to connect to the remote application.  
spring.devtools.remote.proxy.port= # The port of the proxy to use to connect to the remote application.  
spring.devtools.remote.restart.enabled=true # Whether to enable remote restart.  
spring.devtools.remote.secret= # A shared secret required to establish a connection (required to enable remote support).  
spring.devtools.remote.secret-header-name=X-AUTH-TOKEN # HTTP header used to transfer the shared secret.
```



# Properties (Testing Properties)

18 December 2023

01:09 PM

```
# -----  
# TESTING PROPERTIES  
# -----
```

```
spring.test.database.replace=any # Type of existing DataSource to replace.  
spring.test.mockmvc.print=default # MVC Print option.
```

# Day3(Spring Boot JDBC Example)

19 December 2023 10:11 AM

Create application.yml file and delete application.properties

Write below in application.yml for jdbc

```
spring:
datasource:
url: jdbc:mysql://localhost:3306/springbootdb
username: root
password: root
jpa:
hibernate:
ddl-auto: create-drop
```

If we use application.properties file then write code in this format

```
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=create-drop
```

Create employeeDAO class

```
package com.spring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeDAO {

    @Autowired
    JdbcTemplate jdbcTemplate;
```

```

        @GetMapping("/insert")
        public String insertData() {
            jdbcTemplate.execute("insert into employee
            values(101,'Anand','anand@gmail.com',100000,'pune')");
            return "Data inserted successfully";
        }
    }
}

```

Below is main method

```

package com.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJdbcApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootJdbcApplication.class, args);
    }

}

```

Write code for display, update and delete operations

## Day2(Autowiring in spring)

13 December 2023 10:10 AM

Autowiring Modes:

1. No
2. byName
3. byType
4. Constructor
5. autodetect

```
package com.spring.org;

import org.springframework.beans.factory.annotation.Autowired;

public class Parent {
    private String parentName;
    private Child childObj;

    public Parent() {
        out.println("Default Constructor of parent is called");
    }

    public Parent(String parentName, Child childObj) {
        out.println("Parameterized Constructor of parent is called");
        this.parentName = parentName;
        this.childObj = childObj;
    }

    public String getParentName() {
        out.println("get parentName of parent is called");
        return parentName;
    }

    public void setParentName(String parentName) {
        out.println("set parentName of parent is called");
        this.parentName = parentName;
    }
}
```

```

    }
    public Child getChildObj() {
        out.println("getChildObj of parent is called");

        return childObj;
    }

    public void setChildObj(Child childObj) {
        out.println("setChildObj of parent is called");

        this.childObj = childObj;
    }
    @Override
    public String toString() {
        return "Parent [parentName=" + parentName + ",
        childObj=" + childObj + "]";
    }
}

```

```

package com.spring.org;

public class Child {
    private String childName;

    public Child() {
        out.println("Default constructor of child is
        called");
    }

    public Child(String childName) {
        out.println("Parameterized constructor of
        child is called");
        this.childName = childName;
    }

    public String getChildName() {
        out.println("getChildName of child is
        called");
    }
}

```

```

return childName;
    }

    public void setChildName(String childName) {
        System.out.println("ChildName of child is called");
        this.childName = childName;
    }

    @Override
    public String toString() {
        return "Child [childName=" + childName + "]";
    }
}

```

Main Class:

```

package com.spring.org;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AutoWiringExample {
    public static void main(String[] args) {
        ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
        Parent p1=(Parent)context.getBean("parentBean");
        System.out.println(p1);
    }
}

```

Xml file:

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-

```

beans-3.0.xsd">

```
<beanid="parentBean"class="com.spring.org.Parent"  
autowire="byName">  
<propertyname="parentName"value="Manish  
Sharma"></property>
```

```
</bean>
```

```
<!-- <bean id="parentBean"  
class="com.spring.org.Parent" autowire="constructor">  
    <constructor-arg value="  
Stohia"  
type="String"></constructor-arg> </bean> -->  
<beanid="childObj"class="com.spring.org.Child">  
<!-- <property name="childName" value="Kittu  
Sharma"></property> -->  
<constructor-  
argvalue="sagar"type="String"></constructor-arg>  
  
</bean>  
</beans>
```

## Day2(Annotation Example)

13 December 2023 11:51 AM

```
package com.spring.org;

import
org.springframework.beans.factory.annotation.Required
;

public class Employee {
    private int empId;
    private String name;
    private double salary;

    public Employee() {
        out.println("Default constructor of Employee
is called");
    }

    public Employee(int empId, String name, double
salary) {
        out.println("Parameterized constructor of
Employee is called");

        this.empId = empId;
        this.name = name;
        this.salary = salary;
    }

    public int getEmpId() {
        return empId;
    }

    @Required
    public void setEmpId(int empId) {
        this.empId = empId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
```



```

return "Employee [empId=" + empId + ", name=" + name
+ ", salary=" + salary + "];"
    }
}

```

Main Class:

```

package com.spring.org;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {
    public static void main(String[] args) {
        ApplicationContext context =
        new ClassPathXmlApplicationContext("applicationContext.xml");
        Employee e1 = (Employee) context.getBean("empBean");
        System.out.println(e1);
    }
}

```

Xml file:

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/sc
hema/beans

http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-
context-3.0.xsd">
<context:annotation-config/>
<bean id="empBean" class="com.spring.org.Employee">
<property name="name" value="nilam patil"></property>
</bean>
</beans>

```

## Day2(Annotation AutoWired Example)

13 December 2023 11:52 AM

```
package com.spring.org;

import
org.springframework.beans.factory.annotation.Autowired;

public class Parent {
    private String parentName;
    @Autowired //if we write here then child object of parent
    is not called
    private Child childObj;

    public Parent() {
        out.println("Default Constructor of parent is
called");
    }

    public Parent(String parentName, Child childObj) {
        out.println("Parameterized Constructor of parent is
called");
    }

    this.parentName = parentName;
    this.childObj = childObj;

    }

    public String getParentName() {
        out.println("get parentName of parent is called");
        return parentName;
    }

    public void setParentName(String parentName) {
        out.println("set parentName of parent is called");
    }

    this.parentName = parentName;
    }

    public Child getChildObj() {
        out.println("get childObj of parent is called");
    }

    return childObj;
    }

    //@Autowired //if we write here then child object of
    parent is called
```

```

public void setChildObj(Child childObj) {
    out.println("setChildObj of parent is called");

    this.childObj = childObj;
}
@Override
public String toString() {
    return "Parent [parentName=" + parentName + ", childObj="
+ childObj + "]";
}

}

```

```

package com.spring.org;

public class Child {

    private String childName;

    public Child() {
        out.println("Default constructor of child is
called");
    }

    public Child(String childName) {
        out.println("Parameterized constructor of
child is called");
        this.childName = childName;
    }

    public String getChildName() {
        out.println("getChildName of child is
called");

        return childName;
    }

    public void setChildName(String childName) {
        out.println("setChildName of child is

```

```

called");
this.childName = childName;
    }

@Override
public String toString() {
return "Child [childName=" + childName + "];
    }
}

```

```

package com.spring.org;

import
org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlAppli
cationContext;

public class MainClass {
public static void main(String[] args) {
    ApplicationContext context
    ClassPathXmlApplicationContext("applicationContext.xml
1");
    p1 = (Parent) context.getBean("parentBean");
    out.println(p1);
    }
}

```

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/be
ans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/
context"

```

xsi:schemaLocation="http://www.springframework.org/schema/beans

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

<context:annotation-config/>

```
<!-- <bean id="parentBean"
class="com.spring.org.Parent" autowire="byName">
    <property name="parentName">
        <value>parentName</value>
    </property>
</bean> <bean id="childObj"
class="com.spring.org.Child"> <property
name="childName" value="nilam"></property>
</bean> -->
```

```
<bean id="parentBean" class="com.spring.org.Parent">
<property name="parentName" value="abc"></property>
</bean>
```

```
<bean id="childObj" class="com.spring.org.Child">
<property name="childName" value="abc"></property>
</bean>
```

```
</beans>
```

## Day2(Qualifier Example)

14 December 2023 09:39 AM

```
package com.spring.org;

public class Student {
    private String studentName;
    private int age;
    private String div;
    private String schoolName;

    public Student() {
        super();
    }

    public Student(String studentName, int age, String
div, String schoolName) {
        super();
        this.studentName = studentName;
        this.age = age;
        this.div = div;
        this.schoolName = schoolName;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getDiv() {
        return div;
    }
}
```

```

public void setDiv(String div) {
    this.div = div;
}

public String getSchoolName() {
    return schoolName;
}

public void setSchoolName(String schoolName) {
    this.schoolName = schoolName;
}

@Override
public String toString() {
    return "Student [studentName=" + studentName + ",
age=" + age + ", div=" + div + ", schoolName=" +
schoolName
    "]" +
}
}

```

```

package com.spring.org;

import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Qualifier;

public class Profile {
    @Autowired
    @Qualifier("std1")
    private Student std;

    public Profile() {

    }

    public Profile(Student std) {

```

```

super();
this.std = std;
    }

public Student getStd() {
return std;
    }

public void setStd(Student std) {
this.std = std;
    }

@Override
public String toString() {
return "Profile [std=" + std + "]";
    }
}

```

```

package com.spring.org;

import
org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicati
onContext;

public class MainClass {

public static void main(String[] args) {
    ApplicationConte
    ApplicationContext
    ClassPathXmlApplicationContext("applicationContext.xml
    l");
    p1 = (Profile)
context.getBean("profileBean");
    out.println(p1);
    }

}

```



```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/sc
hema/beans
```

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>>

```
<context:annotation-config/>
```

```
<beanid="std1"class="com.spring.org.Student">
<propertyname="studentName"value="Eshan"></property>
<propertyname="age"value="10"></property>
<propertyname="div"value="2std"></property>
<propertyname="schoolName"value="Euro
school"></property>
</bean>
<beanid="std2"class="com.spring.org.Student">
<propertyname="studentName"value="Anish"></property>
<propertyname="age"value="17"></property>
<propertyname="div"value="12std"></property>
<propertyname="schoolName"value="Euro
school"></property>
</bean>
<beanid="profileBean"class="com.spring.org.Profile"><
/bean>
</beans>
```

# Day1

12 December 2023 11:17 AM

## 1. Create employee class

```
public class Employee {
    private int empld;
    private String name;
    private double salary;

    public Employee() {
        System.out.println("Default constructor of Employee is called");
    }

    public Employee(int empld, String name, double salary) {
        System.out.println("Parameterised constructor of Employee is called");
        this.empld = empld;
        this.name = name;
        this.salary = salary;
    }

    public int getEmpld() {
        System.out.println("getEmpld of Employee is called");
        return empld;
    }

    public void setEmpld(int empld) {
        System.out.println("setEmpld of Employee is called");
        this.empld = empld;
    }

    public String getName() {
        System.out.println("getName of Employee is called");
        return name;
    }

    public void setName(String name) {
        System.out.println("setName of Employee is called");
        this.name = name;
    }

    public double getSalary() {
        System.out.println("getSalary of Employee is called");
        return salary;
    }

    public void setSalary(double salary) {
        System.out.println("setSalary of Employee is called");
        this.salary = salary;
    }
}
```

## 2.create xml file under src

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

    <bean id="employeeBean" class="com.training.org.Employee">
        <constructor-arg value="101" type="integer"></constructor-arg>
        <constructor-arg value="EShan MAdake" type="String"></constructor-arg>
        <constructor-arg value="10000" type="double"></constructor-arg>
    </bean>
```

</beans>

### #\$\$. IoC container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:  
to instantiate the application class  
to configure the object  
to assemble the dependencies between the objects  
There are two types of IoC containers. They are:  
BeanFactory  
ApplicationContext

### 3. Create employeemanagement class

```
package com.training.org;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class EmployeeManagement {
    public static void main(String[] args) {

        //1st way to create bean factory
        Resource resource=new ClassPathResource("applicationContext.xml");
        BeanFactory factory=new XmlBeanFactory(resource);

        Employee e1=(Employee) factory.getBean("employeeBean");
        System.out.println(e1);

    }
}
```

### Difference btn BeanFactory and ApplicationContext:

The org.springframework.beans.factory.BeanFactory and the org.springframework.context.ApplicationContext interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

### XmlBeanFactory:

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

```
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

ClassPathXmlApplicationContext:

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("applicationContext.xml");
```

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

bean class:

The bean class is the actual implementation of the bean being described by the BeanFactory.  
Bean examples – DAO, DataSource, Transaction Manager, Persistence Managers, Service objects, etc  
Spring config contains implementation classes while your code should program to interfaces.  
Bean behaviors include:  
Singleton or prototype  
Autowiring  
Initialization and destruction methods  
init-method  
destroy-method  
Beans can be configured to have property values set.  
Can read simple values, collections, maps, references to other beans, etc.

### 3. Under same employeemanagement class

```
package com.training.org;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class EmployeeManagement {
    public static void main(String[] args) {

        //1st way to create IOC Container
        // Resource resource=new ClassPathResource("applicationContext.xml");
        // BeanFactory factory=new XmlBeanFactory(resource);

        //2nd Way to create IOC container

        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        Employee e1=(Employee) context.getBean("employeeBean");
        System.out.println(e1);
```

```
}  
}
```

Whole code of day 1

Employee class:

```
package com.spring.org;  
  
import java.util.List;  
  
public class Employee {  
  
    private int empId;  
    private String name;  
    private double salary;  
    private Address perAddress;  
    private List<String> technologyList;  
    private List<Projects> project;  
  
    public Employee() {  
        out.println("Default constructor of employee is called");  
    }  
  
    public Employee(int empId, String name, double salary, Address address, List<String>  
        technologyList,  
        List<Projects> project) {  
        out.println("Parameterised constructor of Employee is called");  
        this.empId = empId;  
        this.name = name;  
        this.salary = salary;  
        this.perAddress = new Address();  
        this.technologyList = technologyList;  
        this.project = project;  
    }  
  
    public int getEmpId() {  
        out.println("Get EmpId of Employee is called");  
        return empId;  
    }  
  
    public void setEmpId(int empId) {  
        out.println("Set EmpId of Employee is called");  
        this.empId = empId;  
    }  
  
    public String getName() {  
        out.println("Get Name of Employee is called");  
        return name;  
    }  
  
    public void setName(String name) {  
        out.println("Set Name of Employee is called");  
        this.name = name;  
    }  
  
    public double getSalary() {  
        out.println("Get Salary of Employee is called");  
        return salary;  
    }  
  
    public void setSalary(double salary) {  
        out.println("Set Salary of Employee is called");  
        this.salary = salary;  
    }  
}
```

```

public Address getPerAddress() {
    out.printSystem.out.println("getPerAddress of employee is called");

    return perAddress;
}

public void setPerAddress(Address perAddress) {
    out.printSystem.out.println("setPerAddress of employee is called");
    this.perAddress = perAddress;
}

public List<String> getTechnologyList() {
    out.printSystem.out.println("getTechnologyList of employee is called");

    return technologyList;
}

public void setTechnologyList(List<String> technologyList) {
    out.printSystem.out.println("setTechnologyList of employee is called");

    this.technologyList = technologyList;
}

public List<Projects> getProjects() {
    out.printSystem.out.println("getProjects of employee is called");

    return project;
}

public void setProjects(List<Projects> projects) {
    out.printSystem.out.println("setProjects of employee is called");

    this.project = projects;
}

@Override
public String toString() {
    return "Employee [empId=" + empId + ", name=" + name + ", salary=" + salary + ",
    perAddress=" + perAddress
    ", technologyList=" + technologyList + ", project=" + project + "]";
}
}

```

Address Class:

```

package com.spring.org;

public class Address {

    private String street;

    private String cityName;
    private String state;
    private String societyName;
    private String flatNo;

    public Address() {
        out.printSystem.out.println("Default constructor of Address called");
    }

    public Address(String street, String cityName, String state, String societyName,
    String flatNo) {
        out.printSystem.out.println("Parametrized constructor of Address called");
        this.street = street;
        this.cityName = cityName;
        this.state = state;
        this.societyName = societyName;
        this.flatNo = flatNo;
    }
}

```

```

public String getStreet() {
    out.print${tgm}tStreet of Address is called");
    return street;
}

public void setStreet(String street) {
    out.print${tgm}tStreet of Address is called");

    this.street = street;
}

public String getCityName() {
    out.print${tgm}tCityName of Address is called");

    return cityName;
}

public void setCityName(String cityName) {
    out.print${tgm}tCityName of Address is called");

    this.cityName = cityName;
}

public String getState() {
    out.print${tgm}tState of Address is called");

    return state;
}

public void setState(String state) {
    out.print${tgm}tState of Address is called");

    this.state = state;
}

public String getSocietyName() {
    out.print${tgm}tSocietyName of Address is called");

    return societyName;
}

public void setSocietyName(String societyName) {
    out.print${tgm}tSocietyName of Address is called");

    this.societyName = societyName;
}

public String getFlatNo() {
    out.print${tgm}tFlatNo of Address is called");

    return flatNo;
}

public void setFlatNo(String flatNo) {
    out.print${tgm}tFlatNo of Address is called");

    this.flatNo = flatNo;
}

@Override
public String toString() {
    return "Address [street=" + street + ", cityName=" + cityName + ", state=" + state +
        ", societyName="
            + societyName + ", flatNo=" + flatNo + "];"
    }
}

```

Project class:

```

package com.spring.org;

public class Projects {
    private String projectName;
    private String projectManager;
    private String duration;

```

```

public String getProjectName() {
return projectName;
}

public void setProjectName(String projectName) {
this.projectName = projectName;
}

public String getProjectManager() {
return projectManager;
}

public void setProjectManager(String projectManager) {
this.projectManager = projectManager;
}

public String getDuration() {
return duration;
}

public void setDuration(String duration) {
this.duration = duration;
}

@Override
public String toString() {
return "Projects [projectName=" + projectName + ", projectManager=" + projectManager
+ ", duration=" + duration
+ "]";
}
}
}

```

Main Class:

```

package com.spring.org;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class EmployeeManagement {
    public static void main(String[] args) {

        // 1st way to create bean factory
        Resource resource = new ClassPathResource("applicationContext.xml");
        BeanFactory factory = new XmlBeanFactory(resource);

        Employee e1 = (Employee) factory.getBean("employeeBean");
        System.out.println(e1);

        // 2nd Way to create IOC container

        ApplicationContext context = new
        ClassPathXmlApplicationContext("applicationContext.xml");

        Employee e2 = (Employee) context.getBean("employeeBean");
        System.out.println(e2);

    }
}

```

XML file=applicationContext.xml



```

<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<beanid="add1"class="com.spring.org.Address">
<propertyname="cityName"value="Pune"></property>
<propertyname="street"value="Abc"></property>
<propertyname="state"value="Maharashtra"></property>
<propertyname="societyName"value="Long Island"></property>
<propertyname="flatNo"value="A-1102"></property>
</bean>

<beanid="project1"class="com.spring.org.Projects">
<propertyname="projectName"value="City_Bank"></property>
<propertyname="projectManager"value="amit Shah"></property>
<propertyname="duration"value="55 days"></property>
</bean>

<beanid="project2"class="com.spring.org.Projects">
<propertyname="projectName"value="Retail_Bank"></property>
<propertyname="projectManager"value="ram babu"></property>
<propertyname="duration"value="456 days"></property>
</bean>

<beanid="project3"class="com.spring.org.Projects">
<propertyname="projectName"value="Corporate_Bank"></property>
<propertyname="projectManager"value="akash saLavi"></property>
<propertyname="duration"value="235 days"></property>
</bean>

<beanid="employeeBean"class="com.spring.org.Employee">
<!-- <constructor-arg value="101" type="int"></constructor-arg> <constructor-arg
Eshan Kashyap type="String"></constructor-arg> <constructor-arg
value="10000"
</constructor-arg type="double">

<propertyname="empId"value="102"></property>
<propertyname="name"value="Eshan Kashyap"></property>
<propertyname="salary"value="10000.0"></property>
<propertyname="perAddress"ref="add1"></property>
<propertyname="technologyList">
<list>
<value>Java</value>
<value>CPP</value>
<value>Angular</value>

</list>
</property>
<propertyname="projects">
<list>
<refbean="project1"></ref>
<refbean="project2"></ref>
<refbean="project3"></ref>

</list>
</property>

</bean>

</beans>

```

## Day3(JDBC Framework)

14 December 2023 11:39 AM

Xml File: applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">

<!-- Initialization for data source -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver"/>
<property name="url" value="jdbc:mysql://localhost:3306/TEST"/>
<property name="username" value="shital"/>
<property name="password" value="shital@123"/>
</bean>

<!-- Definition for studentJdbcTemplate bean -->
<bean id="studentJdbcTemplate"
class="com.training.org.StudentJdbcTemplate">
<property name="dataSource" ref="dataSource"/>
</bean>
</beans>
```