CS544
Enterprise Application Architecture

Lesson 13 – Scheduling, Events & Configuration

Frameworks and Best Practices Used in Designing Large-Scale Software Systems

Payman Salek, M.S.

Original Material: Prof. Rene de Jong – July 2022

JOB SCHEDULING

Job scheduling

- JDK Timer: java.util.Timer
  - Basic scheduling support
    - Execute at a given time
    - Execute at some fixed frequency
- Quartz scheduling
  - Open source framework
  - Powerful job scheduling engine
  - Cron-based scheduling

Scheduling basics

- Job
  - Unit of work that needs to execute at a specific time or interval
- Trigger
  - The condition (specific time or interval) that causes a job to run
- Schedule
  - A collection of triggers

JDK Timer example

```
public class HelloWorldTask extends TimerTask{

    public void run() {
        Date date =  Calendar.getInstance().getTime();
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
        String currenttime = timeFormatter.format(date);

        System.out.println("This task runs at "+currenttime);
    }
}
```

```
import java.util.Timer;

public class Application {

    public static void main(String[] args) {
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(new HelloWorldTask(), 5000, 5000);
    }
}
```

The job to run

Start the job after 5 seconds

Run the job every 5 seconds

```
This task runs at 10:45:52
This task runs at 10:45:57
This task runs at 10:46:02
This task runs at 10:46:07
```

Quartz cron scheduling example

```
public class HelloWorldJob implements Job{

    public void execute(JobExecutionContext arg0) throws JobExecutionException {
       Date date =  Calendar.getInstance().getTime();
       DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
       String currenttime = timeFormatter.format(date);

       System.out.println("This task runs at "+currenttime);
    }
}
```

```
public class CronApplication {
  public static void main(String[] args) throws SchedulerException, ParseException {
    Scheduler scheduler =new StdSchedulerFactory().getScheduler();
    scheduler.start();

    JobDetail jobDetail=new
             JobDetail("HelloWorldJob",scheduler.DEFAULT_GROUP,HelloWorldJob.class);

    String cronExpression ="0/5 * * * * ?";
    Trigger crontrigger=new
                  CronTrigger("crontrigger",scheduler.DEFAULT_GROUP,cronExpression);
    scheduler.scheduleJob(jobDetail, crontrigger);
  }
}
```

The trigger is expressed with a cron expression

```
This task runs at 12:04:25
This task runs at 12:04:30
This task runs at 12:04:35
```

Quartz cron expressions

- String with 6 or 7 space separated sub-expressions with the following meaning:

  seconds minutes hours dayOfMonth month dayOfWeek year(optional)

- Examples
  - *"0 0 12 ? * WED"*
    - every Wednesday at 12:00 pm
  - "0 0/5 * * * ?"
    - every 5 minutes
  - "10 0/5 * * * ?"
    - every 5 minutes, at 10 seconds after the minute (i.e. 10:00:10 am, 10:05:10 am, etc.).
  - "0 30 10-13 ? * WED,FRI"
    - 10:30, 11:30, 12:30, and 13:30, on every Wednesday and Friday.
  - "0 0/30 8-9 5,20 * ?"
    - every half hour between the hours of 8 am and 10 am on the 5th and 20th of every month.

*=every

?=no specific value (only for dayOfMonth and dayOfWeek)

Spring annotation based scheduling

```
@SpringBootApplication
@EnableScheduling
public class SpringBootSchedulingApplication {

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

Enable scheduling

```
@Component
public class WelcomeTask {

  @Scheduled(fixedRate = 5000)
  public void welcome() {
    Date date = Calendar.getInstance().getTime();
    DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
    String currenttime = timeFormatter.format(date);
    System.out.println("This task runs at " + currenttime);
  }
}
```

Run every 5 seconds

```
This task runs at 12:07:50
This task runs at 12:07:55
This task runs at 12:08:00
```

@Scheduled

```
@Scheduled(fixedDelay = 5000)
public void welcome() {
```

Run every 5 seconds measured from the completion time of the welcome() method

```
@Scheduled(fixedRate = 5000)
public void welcome() {
```

Run every 5 seconds measured from the start time of the welcome() method

```
@Scheduled(initialDelay=1000, fixedRate=5000)
public void welcome() {
```

Run every 5 seconds but wait 1 second before the first execution

```
@Scheduled(cron="*/5 * * * * MON-FRI")
public void welcome() {
```

Cron expression: Run every 5 seconds on Monday till Friday.

Main point

- Spring makes it simple to schedule methods of spring beans.

  Science of Consciousness: There is order in creation. In creation everything happens according the laws of Nature.

EVENTS
ASYNCHRONOUS METHODS

Events

```
public class AddCustomerEvent {
  private String message;

  public AddCustomerEvent(String message) {
    this.message = message;
  }

  public String getMessage() {
    return message;
  }
}
```

A simple event class

Event publisher and listener

```java
@Service
public class CustomerServiceImpl implements CustomerService {
  @Autowired
  private ApplicationEventPublisher publisher;

  public void addCustomer() {
    publisher.publishEvent(new AddCustomerEvent("New customer is added"));
  }
}
```

Inject a publisher

```java
@Service
public class Listener {

  @EventListener
  public void onEvent(AddCustomerEvent event) {
    System.out.println("received event :" + event.getMessage());;
  }
}
```

Listen to AddCustomer events

Asynchronous events

```
@Service
@EnableAsync
public class Listener {

  @Async
  @EventListener
  public void onEvent(AddCustomerEvent event) {
    System.out.println("received event :" + event.getMessage());;
  }
}
```

Asynchronous methods

```
@EnableAsync
public class MyServiceImpl implements MyService {
  @Async
  public void welcome() {
    Date date = Calendar.getInstance().getTime();
    DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
    String currenttime = timeFormatter.format(date);
    System.out.println("This task runs at " + currenttime);
}
```

Enable asynchronous methods

The method call returns immediately

Main point

- Spring events is a powerful technique to implement publish subscribe within the application.

  Science of Consciousness: When one subscribes daily to the intelligence of nature one automatically receives support of Nature.

SPRING BOOT CONFIGURATION

@Value

```java
@Service
public class EmailServiceImpl implements EmailService{
  @Value(" ${smtpserver}")
  String outgoingMailServer;

  ...
}
```

Works for small and simple data, not for complex data

```
smtpserver=smtp.mydomain.com
```

@ConfigurationProperties

application.properties

```
Mapping single properties
myapp.mail.to=frank@hotmail.com
myapp.mail.host=mail.example.com
myapp.mail.port=250

#Mapping list or array
myapp.mail.cc=mike@gmail.com,david@gmail.com
myapp.mail.bcc=john@hotmail.com,admin@acme.com

#Mapping nested POJO class
myapp.mail.credential.user-name=john1234
myapp.mail.credential.password=xyz@1234
```

application.yml

```
myapp:
  mail:
    to: frank@hotmail.com
    host: mail.example.com
    port: 250
    cc:
      - mike@gmail.com
      - david@gmail.com
    bcc:
      - john@hotmail.com
      - admin@acme.com
    credential:
      user-name: john1234
      password: xyz@1234
```

@ConfigurationProperties

```java
@ConfigurationProperties(prefix="myapp.mail")
public class MailProperties {

  private String to;
  private String host;
  private int port;
  private String[] cc;
  private List<String> bcc;

  private Credential credential = new Credential();

  //Setter and Getter methods

  public class Credential {
    private String userName;
    private String password;
    //Setter and Getter methods

  }
}
```

```
Mapping single properties
myapp.mail.to=frank@hotmail.com
myapp.mail.host=mail.example.com
myapp.mail.port=250

#Mapping list or array
myapp.mail.cc=mike@gmail.com,david@gmail.com
myapp.mail.bcc=john@hotmail.com,admin@acme.com

#Mapping nested POJO class
myapp.mail.credential.user-name=john1234
myapp.mail.credential.password=xyz@1234
```

Using MailProperties

```
@Component
public class MailService {

  @Autowired
  private MailProperties mailProperties;

  public void print() {
    System.out.println("Mail TO = " + mailProperties.getTo());
    System.out.println("HOST = " + mailProperties.getHost());
    System.out.println("PORT = " + mailProperties.getPort());
    System.out.println();

    //Print list or array
    System.out.println("Mail CC = " + String.join(", ", mailProperties.getCc()));
    System.out.println("Mail BCC = " + mailProperties.getBcc());
    System.out.println();

    //Print nested bean's properties
    System.out.println("User Name = " + mailProperties.getCredential().getUserName());
    System.out.println("Password = " + mailProperties.getCredential().getPassword());
  }
}
```

Using MailProperties

```
@SpringBootApplication
@EnableConfigurationProperties(MailProperties.class)
public class SpringBootProjectApplication implements CommandLineRunner {
  @Autowired
  private MailService mailService;

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

  @Override
  public void run(String... args) throws Exception {
    mailService.print();
  }
}
```

@EnableConfigurationProperties

Advantages of @ConfigurationProperties

- Relaxed binding
  - If the property is **db.username**
  - Then these all will work:
    - db.user-name
    - db.user_name
    - db.UserName
- Property validation
  - Properties van be validated using JSR-303 validation annotation

Property Validation

```
@ConfigurationProperties(prefix="myapp.mail")
@Validated
public class MailProperties {
  @Email
  private String to;
  @NotBlank
  private String host;
  private int port;
  private String[] cc;
  private List<String> bcc;

  private Credential credential = new Credential();

  //Setter and Getter methods

  @Valid
  public class Credential {
    @NotBlank
    private String userName;
    @Size(max = 15, min = 6)
    private String password;

    //Setter and Getter methods

  }
}
```

Property Validation

```yaml
myapp:
  mail:
    to: frankhotmail.com
    host:
    port: 250
    cc:
      - mike@gmail.com
      - david@gmail.com
    bcc:
      - john@hotmail.com
      - admin@acme.com
    credential:
      user-name: john1234
      password: xyz@1234
```

```xml
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.0.5.Final</version>
</dependency>
```

```
Binding to target org.springframework.boot.context.properties.bind.BindException:

    Property: myapp.mail.to
    Value: frankhotmail.com
    Origin: class path resource [application.yml]:3:9
    Reason: must be a well-formed email address

    Property: myapp.mail.host
    Value:
    Origin: class path resource [application.yml]:4:10
    Reason: must not be blank
```

Configure a different webserver

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

Spring Boot starts Tomcat by default

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

Start the embedded jetty webserver

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>
```

Undertow webserver

Spring Boot Properties

- https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

```
# ----------------------------------------
# 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-base

# 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.
```

Main point

- @ConfigurationProperties allows to package configuration properties together and in addition provides property validation.

  Science of Consciousness: In cosmic consciousness one spontaneously handles according the laws of Nature.