# 19-Scheduled Task



## Learning Objectives

---

> Understand the usage of Data Transfer Object (DTO) and its benefits
>
> How to use it in Spring Boot Project.
>
> What is ModelMapper and how convenient it is.
>
> Design Mapper Class & it's Methods, converting between DTO, Domain Object & Entity.
>
> Three types of Mapper Class - MapStruct, ModelMapper, Custom Mapper

## Introduction

In Spring Boot, you can schedule tasks to run at fixed intervals or specific times using the `@Scheduled` annotation, which is part of the Spring Framework's scheduling support. This feature allows you to define methods in your Spring Boot application that should be executed at regular intervals or based on a cron expression.

# Enable Scheduling

To enable scheduling in your Spring Boot application, you need to annotate your main application class (the one with the `public static void main` method) with `@EnableScheduling`. This tells Spring to scan for and execute scheduled tasks.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling // inject a specific into the context
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

# Create Scheduled Tasks

To create a scheduled task, you can annotate a method with `@Scheduled`. This annotation provides various attributes to specify the scheduling details, such as fixed delay, fixed rate, and cron expressions.

## 1. Fixed Delay

Execute the task with a fixed delay between the end of the last execution and the start of the next execution (in milliseconds).

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class MyScheduledTask {

    @Scheduled(fixedDelay = 5000) // Run every 5 seconds
    public void runTask() {
        // Your task logic here
    }
}
```

## 2. Fixed Rate

Execute the task at a fixed rate (in milliseconds), regardless of how long the task takes to complete.

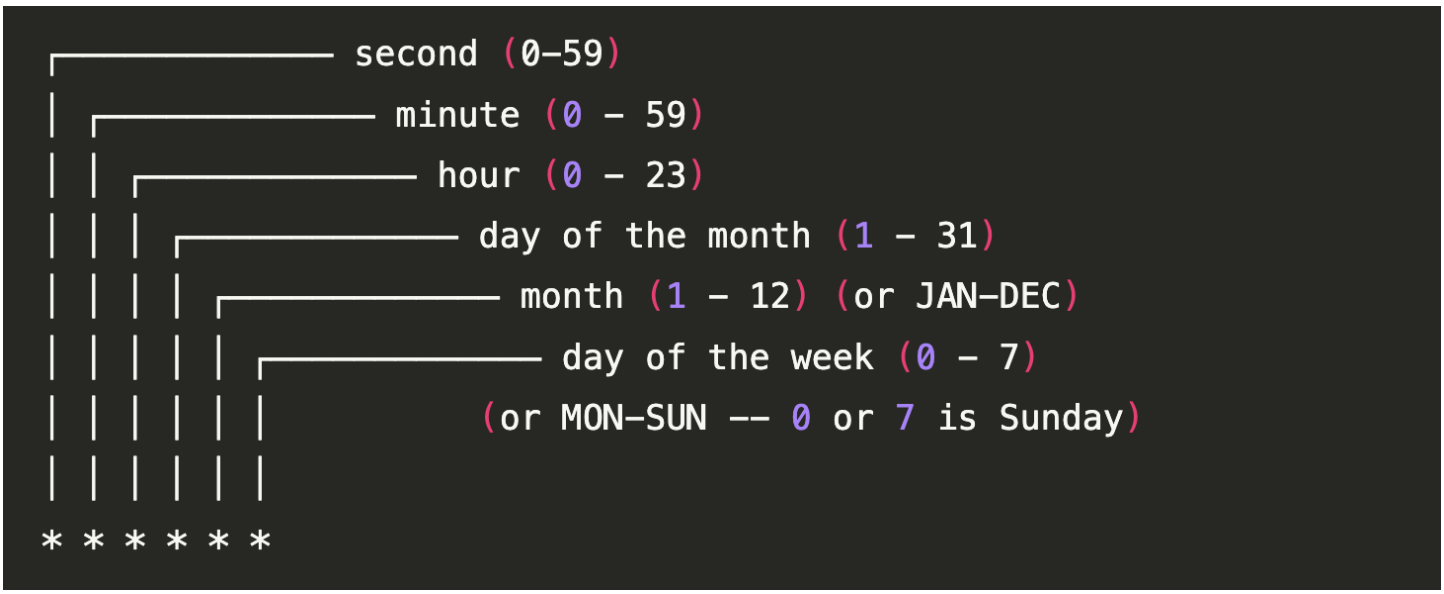
```java
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class MyScheduledTask {

    @Scheduled(fixedRate = 60000) // Run every 60 seconds
    public void runTask() {
        // Your task logic here
    }
}
```

## 3. Cron Expression

A cron expression is a string representing a schedule in a human-readable format. It is commonly used in various scheduling contexts, such as in Unix/Linux cron jobs and in scheduling tasks in programming languages like Java (e.g., with Spring's `@Scheduled` annotation). A cron expression consists of several fields, each representing a unit of time, and it specifies when a task or job should run. The format of a cron expression may vary slightly depending on the system or library used, but the general structure remains consistent.

Here is a detailed breakdown of the cron expression format:

### 3.1 Field Order




A typical cron expression is divided into five or six fields, representing the following units of time:

Field 1: Seconds (0 - 59)

Field 2: Minutes (0 - 59)

Field 3: Hours (0 - 23)

Field 4: Day of the Month (1 - 31)

Field 5: Month (1 - 12 or JAN - DEC)

Field 6: Day of the Week (0 - 7 or SUN - SAT, where both 0 and 7 represent Sunday)

## 3.2 Wildcard ( `*` )

An asterisk `*` is used to represent "every" or "any" value for a field. For example, `*` in the minutes field means "every minute."

## 3.3 Specific Values

You can specify a single value for a field. For example, `5` in the hours field means "at the 5th hour."

## 3.4 Ranges

You can specify a range of values for a field using a hyphen `-`. For example, `1-5` in the day of the week field means "from Monday to Friday."

## 3.5 Lists

You can specify a list of values for a field using a comma `,`. For example, `1,15` in the day of the month field means "on the 1st and 15th day of the month."

## 3.6 Step Values

You can use a forward slash `/` to specify step values. For example, `*/2` in the minutes field means "every 2 minutes."

## 3.7 Named Abbreviations

Some cron implementations support named abbreviations for months and days of the week. For example, `JAN` or `SUN` can be used to represent January and Sunday, respectively.

## 3.8 Last Day of the Month

You can use `L` to represent the last day of the month in the day of the month field. For example, `L` in the day of the month field means "on the last day of the month."

## 3.9 Last Weekday of the Month

Some cron implementations support `LW` to represent the last weekday of the month. For example, `LW` in the day of the month field means "on the last weekday of the month."

### 3.10 Hash (#)

In some cron systems, you can use a `#` to specify the `N` th occurrence of a weekday in a month. For example, `2#3` in the day of the week field means "the third Tuesday of the month."

### 3.11 Time Zone

Some cron systems support specifying the time zone as an optional field, typically at the beginning of the expression. For example, `TZ=America/New_York` to set the time zone to Eastern Time.

### Examples of Cron Expressions

- `"0 0 * * * *"` : Run every hour at the beginning of the hour.
- `"0 0 0 * * *"` : Run every day at midnight.
- `"0 0 12,18 * * MON-FRI"` : Run every weekday at 12 PM and 6 PM.
- `"0 0 1 * * ?"` : Run on the first day of every month.
- `"0 15 10 ? * 5#3"` : Run on the third Friday of every month at 10:15 AM.

Please note that the format and supported features of cron expressions may vary depending on the system or library you are using. It's essential to consult the documentation of the specific scheduler or library you are working with to understand the exact cron expression format it supports.

```
1  import org.springframework.scheduling.annotation.Scheduled;
2  import org.springframework.stereotype.Component;
3
4  @Component
5  public class MyScheduledTask {
6
7      @Scheduled(cron = "0 0 12,18 * * MON-FRI") // Run every weekday at 12 PM
   and 6 PM.
8      public void runTask() {
9          // Your task logic here
10     }
11 }
```

## 4. Customizing Task Scheduling

You can further customize the scheduling behavior by using attributes like `initialDelay` to specify a delay before the first execution, and `zone` to set the time zone for cron expressions.

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class MyScheduledTask {

    // Run on the third Friday of every month at 10:15 AM. in zone
Europe/London
    @Scheduled(cron = "0 15 10 ? * 6#3", zone = "Europe/London"
    public void scheduledTask() {
        // Your task logic here
    }

}
```

Your scheduled task will run based on the specified scheduling configuration. You can create multiple scheduled tasks in your application by adding more methods annotated with `@Scheduled`.

Scheduled tasks are particularly useful for executing background jobs, recurring tasks, and periodic maintenance in your Spring Boot applications.

## Note

The format and supported features of cron expressions may vary depending on the system or library you are using. It's essential to consult the documentation of the specific scheduler or library you are working with to understand the exact cron expression format it supports.