

第六章 Spring Boot 定时任务

(SpringBoot 高级)

课程内容

- Scheduled 定时任务器
- 整合 Quartz 定时任务框架

一、Scheduled 定时任务器

Scheduled 定时任务器：是 Spring3.0 以后自带的一个定时任务器。

1 在 pom 文件中添加 Scheduled 的坐标

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.RELEASE</version>
    </parent>
    <groupId>com.bjsxt</groupId>
    <artifactId>25-spring-boot-scheduled</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <java.version>1.7</java.version>
        <thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>

        <thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.version>
    </properties>

    <dependencies>
        <!-- springBoot 的启动器 -->
        <dependency>
```

```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- springBoot 的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- 添加 Scheduled 坐标 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>
</dependencies>
</project>

```

2 编写定时任务类

```

/**
 * Scheduled 定时任务
 *
 *
 */
@Component
public class ScheduledDemo {

    /**
     * 定时任务方法
     * @Scheduled: 设置定时任务
     * cron 属性: cron 表达式。定时任务触发是时间的一个字符串表达形式
     */
    @Scheduled(cron="0/2 * * * * ?")
    public void scheduledMethod(){
        System.out.println("定时器被触发"+new Date());
    }
}

```

3 在启动类中开启定时任务的使用

```

/**
 *
 *Scheduled

```

```

    *
    */
    @SpringBootApplication
    @EnableScheduling
    public class App {

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

```

4 cron 表达式讲解

Cron 表达式是一个字符串，分为 6 或 7 个域，每一个域代表一个含义

Cron 有如下两种语法格式：

(1) Seconds Minutes Hours Day Month Week Year

(2) Seconds Minutes Hours Day Month Week

一、结构

corn 从左到右（用空格隔开）：秒 分 小时 月份中的日期 月份 星期中的日期 年份

二、各字段的含义

位置	时间域名	允许值	允许的特殊字符
1	秒	0-59	, - * /
2	分钟	0-59	, - * /
3	小时	0-23	, - * /
4	日	1-31	, - * / L W C
5	月	1-12	, - * /
6	星期	1-7	, - * ? / L C #
7	年(可选)	1970-2099	, - * /

Cron 表达式的时间字段除允许设置数值外，还可使用一些特殊的字符，提供列表、范围、通配符等功能，细说如下：

●星号(*): 可用在所有字段中, 表示对应时间域的每一个时刻, 例如, *在分钟字段时, 表示“每分钟”;
●问号(?): 该字符只在日期和星期字段中使用, 它通常指定为“无意义的值”, 相当于占位符;
●减号(-): 表达一个范围, 如在小时字段中使用“10-12”, 则表示从 10 到 12 点, 即 10,11,12;
●逗号(,): 表达一个列表值, 如在星期字段中使用“MON,WED,FRI”, 则表示星期一, 星期三和星期五;

●斜杠(/): x/y 表达一个等步长序列, x 为起始值, y 为增量步长值。如在分钟字段中使用 0/15, 则表示为 0,15,30 和 45 秒, 而 5/15 在分钟字段中表示 5,20,35,50, 你也可以使用*/y, 它等同于 0/y;

●L: 该字符只在日期和星期字段中使用, 代表“Last”的意思, 但它在两个字段中意思不同。L 在日期字段中, 表示这个月份的最后一天, 如一月的 31 号, 非闰年二月的 28 号; 如果 L 用在星期中, 则表示星期六, 等同于 7。但是, 如果 L 出现在星期字段里, 而且在前面有一个数值 X, 则表示“这个月的最后 X 天”, 例如, 6L 表示该月的最后星期五;

●W: 该字符只能出现在日期字段里, 是对前导日期的修饰, 表示离该日期最近的工作日。例如 15W 表示离该月 15 号最近的工作日, 如果该月 15 号是星期六, 则匹配 14 号星期五; 如果 15 日是星期日, 则匹配 16 号星期一; 如果 15 号是星期二, 那结果就是 15 号星期二。但必须注意关联的匹配日期不能够跨月, 如你指定 1W, 如果 1 号是星期六, 结果匹配的是 3 号星期一, 而非上个月最后的那天。W 字符串只能指定单一日期, 而不能指定日期范围;

●LW 组合: 在日期字段可以组合使用 LW, 它的意思是当月的最后一个工作日;

●井号(#): 该字符只能在星期字段中使用, 表示当月某个工作日。如 6#3 表示当月的第三个星期五(6 表示星期五, #3 表示当前的第三个), 而 4#5 表示当月的第五个星期三, 假设当月没有第五个星期三, 忽略不触发;

●C: 该字符只在日期和星期字段中使用, 代表“Calendar”的意思。它的意思是计划所关联的日期, 如果日期没有被关联, 则相当于日历中所有日期。例如 5C 在日期字段中就相当于日历 5 日以后的第一天。1C 在星期字段中相当于星期日后的第一天。

Cron 表达式对特殊字符的大小写不敏感, 对代表星期的缩写英文大小写也不敏感。

例子:

@Scheduled(cron = "0 0 1 1 1 ?")//每年一月的一号的 1:00:00 执行一次

@Scheduled(cron = "0 0 1 1 1,6 ?") //一月和六月的一号的 1:00:00 执行一次

@Scheduled(cron = "0 0 1 1 1,4,7,10 ?") //每个季度的第一个月的一号的 1:00:00 执行一次

@Scheduled(cron = "0 0 1 1 * ?")//每月一号 1:00:00 执行一次

@Scheduled(cron="0 0 1 * * *") //每天凌晨 1 点执行一次

二、 Spring Boot 整合 Quartz 定时任务框架

1 Quartz 的介绍以及 Quartz 的使用思路

1.1 Quartz 的介绍

quartz (开源项目)

编辑

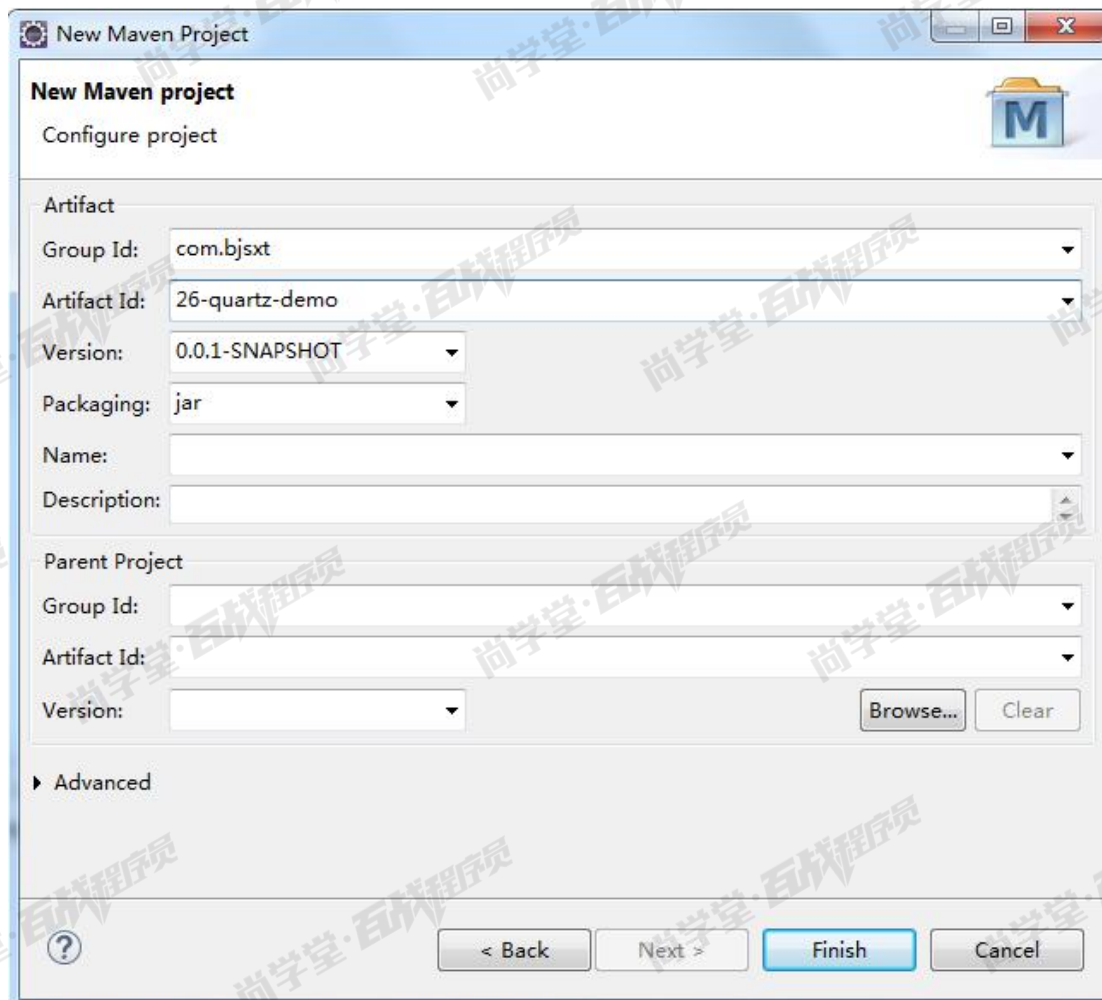
Quartz是OpenSymphony开源组织在Job scheduling领域又一个开源项目，它可以与J2EE与J2SE应用程序相结合也可以单独使用。Quartz可以用来创建简单或为运行十个，百个，甚至是好几万个Jobs这样复杂的程序。Jobs可以做成标准的Java组件或EJBs。Quartz的最新版本为Quartz 2.3.0。

1.2 Quartz 的使用思路

- 1) job - 任务 - 你要做什么事？
- 2) Trigger - 触发器 - 你什么时候去做？
- 3) Scheduler - 任务调度 - 你什么时候需要去做什么事？

2 Quartz 的基本使用方式

2.1 创建项目



New Maven Project

Configure project

Artifact

Group Id: com.bjsxt

Artifact Id: 26-quartz-demo

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

2.2 修改 pom 文件添加 Quartz 的坐标

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.bjsxt</groupId>
  <artifactId>26-quartz-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <!-- Quartz 坐标 -->
```



```

<dependency>
  <groupId>org.quartz-scheduler</groupId>
  <artifactId>quartz</artifactId>
  <version>2.2.1</version>
</dependency>
</dependencies>
</project>

```

2.3 创建 Job 类

```

/**
 * 定义任务类
 *
 */
public class QuartzDemo implements Job {

    /**
     * 任务被触发时所执行的方法
     */
    public void execute(JobExecutionContext arg0) throws
    JobExecutionException {
        System.out.println("Execute..." + new Date());
    }

}

```

2.4 编写测试代码

```

public class QuartzMain {

    public static void main(String[] args) throws Exception {

        // 1. 创建 Job 对象: 你要做什么事?
        JobDetail job = JobBuilder.newJob(QuartzDemo.class).build();

        /**
         * 简单的 trigger 触发时间: 通过 Quartz 提供一个方法来完成简单的重复
调用 cron
         * Trigger: 按照 Cron 的表达式来给定触发的时间
         */
        // 2. 创建 Trigger 对象: 在什么时间做?
        /*Trigger trigger =

```

```

TriggerBuilder.newTrigger().withSchedule(SimpleScheduleBuilder.repeatSe-
condlyForever())

        .build();*/

        Trigger trigger =
TriggerBuilder.newTrigger().withSchedule(CronScheduleBuilder.cronSchedu-
le("0/2 * * * * ?"))
        .build();

        // 3.创建 Scheduler 对象：在什么时间做什么事？
        Scheduler scheduler =
StdSchedulerFactory.getDefaultScheduler();
        scheduler.scheduleJob(job, trigger);

        //启动
        scheduler.start();
    }
}

```

3 .Spring Boot 整合 Quartz 定时框架

3.1 修改 pom 文件添加坐标

```

<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.RELEASE</version>
    </parent>
    <groupId>com.bjsxt</groupId>
    <artifactId>27-spring-boot-quartz</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>

```



```
<java.version>1.7</java.version>
<thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>

<thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.version>
</properties>

<dependencies>
    <!-- springBoot 的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- springBoot 的启动器 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- Quartz 坐标 -->
    <dependency>
        <groupId>org.quartz-scheduler</groupId>
        <artifactId>quartz</artifactId>
        <version>2.2.1</version>
        <exclusions>
            <exclusion>
                <artifactId>slf4j-api</artifactId>
                <groupId>org.slf4j</groupId>
            </exclusion>
        </exclusions>
    </dependency>

    <!-- 添加 Scheduled 坐标 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>

    <!-- Spring tx 坐标 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>
</dependencies>
```

```
</project>
```

3.2 编写 Quartz 的启动类

```
/**
 * Quartz 配置类
 *
 */
@Configuration
public class QuartzConfig {

    /**
     * 1.创建 Job 对象
     */
    @Bean
    public JobDetailFactoryBean jobDetailFactoryBean(){
        JobDetailFactoryBean factory = new JobDetailFactoryBean();
        //关联我们自己的 Job 类
        factory.setJobClass(QuartzDemo.class);
        return factory;
    }

    /**
     * 2.创建 Trigger 对象
     * 简单的 Trigger
     */
    @Bean
    public SimpleTriggerFactoryBean
    simpleTriggerFactoryBean(JobDetailFactoryBean jobDetailFactoryBean){
        SimpleTriggerFactoryBean factory = new
        SimpleTriggerFactoryBean();
        //关联 JobDetail 对象
        factory.setJobDetail(jobDetailFactoryBean.getObject());
        //该参数表示一个执行的毫秒数
        factory.setRepeatInterval(2000);
        //重复次数
        factory.setRepeatCount(5);
        return factory;
    }
}
```

```

/**
 * 3.创建 Scheduler 对象
 */
@Bean
public SchedulerFactoryBean
schedulerFactoryBean(SimpleTriggerFactoryBean simpleTriggerFactoryBean){
    SchedulerFactoryBean factory = new SchedulerFactoryBean();
    //关联 trigger
    factory.setTriggers(simpleTriggerFactoryBean.getObject());

    return factory;
}
}

```

3.3 修改启动类

```

/**
 *
 *spring Boot 整合 Quartz 案例
 *
 */
@SpringBootApplication
@EnableScheduling
public class App {

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

```

4 Job 类中注入对象

4.1 注入时会产生异常

```

org.quartz.SchedulerException: Job threw an unhandled exception.
    at org.quartz.core.JobRunShell.run(JobRunShell.java:213) ~[quartz-2.2.1.jar:na]
    at org.quartz.simpl.SimpleThreadPool$WorkerThread.run(SimpleThreadPool.java:573) [c
Caused by: java.lang.NullPointerException: null
    at com.bjsxt.quartz.QuartzDemo.execute(QuartzDemo.java:24) ~[classes/:na]
    at org.quartz.core.JobRunShell.run(JobRunShell.java:202) ~[quartz-2.2.1.jar:na]
    ... 1 common frames omitted

```

4.2 编写一个 MyAdaptableJobFactory 解决该问题

```
@Component("myAdaptableJobFactory")
public class MyAdaptableJobFactory extends AdaptableJobFactory {

    //AutowireCapableBeanFactory 可以将一个对象添加到 SpringIOC 容器中，
    并且完成该对象注入
    @Autowired
    private AutowireCapableBeanFactory autowireCapableBeanFactory;

    /**
     * 该方法需要将实例化的任务对象手动的添加到 springIOC 容器中并且完成对
    象的注入
     */
    @Override
    protected Object createJobInstance(TriggerFiredBundle bundle)
    throws Exception {
        Object obj = super.createJobInstance(bundle);
        //将 obj 对象添加 Spring IOC 容器中，并完成注入
        this.autowireCapableBeanFactory.autowireBean(obj);
        return obj;
    }
}
```

4.3 修改 QuartzConfig 类

```
/**
 * Quartz 配置类
 *
 *
 */
@Configuration
public class QuartzConfig {

    /**
     * 1.创建 Job 对象
     */
    @Bean
    public JobDetailFactoryBean jobDetailFactoryBean(){
        JobDetailFactoryBean factory = new JobDetailFactoryBean();
    }
}
```

```

        //关联我们自己的 Job 类
        factory.setJobClass(QuartzDemo.class);
        return factory;
    }

    /**
     * 2.创建 Trigger 对象
     * 简单的 Trigger
     */
    /*@Bean
    public SimpleTriggerFactoryBean
    simpleTriggerFactoryBean(JobDetailFactoryBean jobDetailFactoryBean){
        SimpleTriggerFactoryBean factory = new
        SimpleTriggerFactoryBean();
        //关联 JobDetail 对象
        factory.setJobDetail(jobDetailFactoryBean.getObject());
        //该参数表示一个执行的毫秒数
        factory.setRepeatInterval(2000);
        //重复次数
        factory.setRepeatCount(5);
        return factory;
    }*/

    /**
     * Cron Trigger
     */
    @Bean
    public CronTriggerFactoryBean
    cronTriggerFactoryBean(JobDetailFactoryBean jobDetailFactoryBean){
        CronTriggerFactoryBean factory = new CronTriggerFactoryBean();
        factory.setJobDetail(jobDetailFactoryBean.getObject());
        //设置触发时间
        factory.setCronExpression("0/2 * * * * ?");
        return factory;
    }

    /**
     * 3.创建 Scheduler 对象
     */
    @Bean
    public SchedulerFactoryBean
    schedulerFactoryBean(CronTriggerFactoryBean
    cronTriggerFactoryBean, MyAdaptableJobFactory myAdaptableJobFactory){
        SchedulerFactoryBean factory = new SchedulerFactoryBean();

```

```
//关联 trigger
factory.setTriggers(cronTriggerFactoryBean.getObject());
factory.setJobFactory(myAdaptableJobFactory);
return factory;
}
```