定时任务| xxl-job

单机定时任务

java 定时任务可以使某段代码在被调用后的指定时间后、某时刻或者每隔一段时间自动执行。

常见场景：

- 经过指定时间后才执行。
- 每隔指定时间都执行一次。
- 每天的某时刻自动执行。

JAVA 原生 Timer

java 原生定时任务通过类 Timer 和类 TimerTask 来实现的。其中 TimerTask 来构建一个任务，Timer 来作为定时器工具。

构建任务

通过 TimerTask 类来构建一个任务，任务逻辑写在 run 方法中：

Java

```java
TimerTask timerTask = new TimerTask() {

    @Override
    public void run() {
        System.out.println("2s后执行一次...");
    }

};
```

定时器设置

使用定时器工具 Timer 来构建定时任务：

Java

```java
// 调用2s后开始执行，仅执行一次
new Timer().schedule(timerTask,2000);
// 调用2s后开始执行，之后每隔3s重复执行
new Timer().schedule(timerTask,2000,3000);
// 调用后到达time指定的时刻后开始执行，之后每3s执行一次
new Timer().schedule(timerTask,time,3000);
```

如果我们想使任务在每天的某时刻自动执行，可以使用 Calendar 来获取该时刻：

Java

```java
// 获取12:37:00的时间对象
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.HOUR_OF_DAY,12); // 时
calendar.set(Calendar.MINUTE,37);      // 分
calendar.set(Calendar.SECOND,0);       // 秒
Date time = calendar.getTime();

// 设置延迟到12:37:00执行，并且每隔24小时执行一次
new Timer().schedule(timerTask,time,1000 * 60 * 60 * 24);
```

实例

每5s打印一次”hello“的定时任务：

Java

```java
public static void main(String[] args){

    TimerTask task = new TimerTask() {
```

```
        @Override
        public void run() {
            System.out.println("hello");
        }
    };
    new Timer().schedule(task,5000,5000);

}
```

注意：启动多个任务时不要使用同一个定时器timer，否则任务间会互相影响。

JAVA 原生 线程池

jdk1.5 中提供定时任务线程池，可以使用定时执行线程的方式实现定时任务：

构建定时任务

实际上就是一个线程：

Java

```
Runnable runnable=()->{
    System.out.println("执行一个任务...");
};
```

构建定时线程

Java

```
// 定时任务线程池，容量为10个线程
ScheduledExecutorService service = Executors.newScheduledThreadPool(10);
// 定时执行线程
service.scheduleAtFixedRate(runnable,1,3, TimeUnit.SECONDS);
```

实例

Java

```
public static void main(String[] args) {

    //定义一个任务线程
    Runnable runnable=()->{
        System.out.println("执行一个任务...");
    };
    // 定时任务线程池，容量为10个线程
    ScheduledExecutorService service = Executors.newScheduledThreadPool(10);
    // 定时执行线程
    service.scheduleAtFixedRate(runnable,1,3, TimeUnit.SECONDS);

}
```

Spring Task

Spring framework 提供定时任务功能，SpringBoot 项目中只需要使用两个注解就可以设置一个定时任务了。

Java

```
@Component
@EnableScheduling  // 开启定时任务
public class TimeMethod {

    @Scheduled(cron = "0/2 * * * * ?")
    public void consumer(){
        System.out.println("执行了一个定时任务...");
    }

}
```

cron 表达式在线生成：https://cron.qqe2.com/

Cron 表达式语法

cron 表达式用空格划分为 7 个部分：

字段序号	字段含义	取值范围	可取字符	备注
1	秒	0–59	, – * /	
2	分	0–59	, – * /	
3	时	0–23	, – * /	
4	日	1–31	, – * / ? L W C	取值范围按照实际月份的天数
5	月	1–12	, – * /	取值也可以是JAN–DEC格式
6	周几	1–7	, – * / ? # L C	取值可以是SUN–SAT （1=SUN）格式
7	年	1970~2099	, – * /	可以不写

特殊字符含义：

字符	含义	备注
a,b	枚举时间段	每次经过a或b时间段后，都会触发
a–b	范围内所有时间段	表示a–b范围内所有时间点
*	所有时间段	表示取值范围内所有时间点
a/b	延时循环	延时a时间后，每隔b时间触发一次
?	每个时间段	通常用在周字段，可以匹配每一天，却不将每天都匹配，可以理解为忽略此字段
a#b	当月第几个周几	第b个周（a–1）
L	最后	时间段快结束时触发
W	有效工作日	周一到周五，LW连用表示最后一个工作日
C		

关于? ： 假设每个月20日触发任务，20日可以是一周的任意一天，如果周字段使用 *，会导致每个月每天都会触发，因为21号会符合周字段条件，因此使用? 即可解决。

quartz

分布式定时任务调度中心

XXL–JOB

调度中心

1.构建数据库

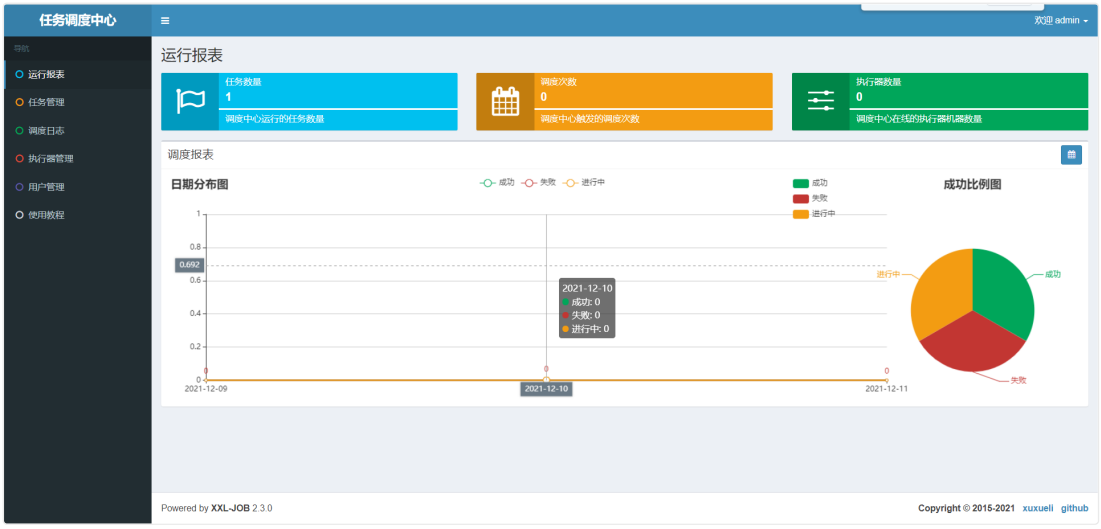
tables_xxl_job.sql 6.72 KB

2.下载xxl-job-admin项目并启动：Gitee （多节点的话，使用同一个库）

- http://localhost:9999/xxl-job-admin/toLogin
- 默认登录账号“admin/123456”

执行器

例子可下载：xxl-job-executor-samples



新建项目并引入maven依赖：

XML

```xml
<dependency>
    <groupId>com.xuxueli</groupId>
    <artifactId>xxl-job-core</artifactId>
    <version>2.1.2</version>
</dependency>
```

yml配置：

YAML

```yaml
xxl:
  job:
    # 调度中心配置
    admin:
      addresses: http://127.0.0.1:9999/xxl-job-admin # 调度中心地址，多个用逗号分隔 。执行器将会使用该地址进行"执行器心跳注册"和"任务结果回调"；为空
    accessToken:  # 执行器通讯TOKEN：非空时启用；
    # 执行器配置
    executor:
      appname: xxl-job-executor-demo
      address:   # 执行器注册地址，默认使用内嵌服务IP:PORT
      ip:        # 执行器IP，默认为空表示自动获取IP
      port: 8088 # 执行器端口，<=0则自动获取；默认9999，单机部署多个执行器时，要配置不同端口
      logpath: /data/applogs/xxl-job/jobhandler # 日志文件路径
      logretentiondays: 30 # 日志保存天数，>=3生效
```

使用配置类：

Java

```java
@Configuration
public class JobConfig {
    private Logger logger = LoggerFactory.getLogger(JobConfig.class);

    @Value("${xxl.job.admin.addresses}")
    private String adminAddresses;

    @Value("${xxl.job.accessToken}")
    private String accessToken;

    @Value("${xxl.job.executor.appname}")
    private String appname;

    @Value("${xxl.job.executor.address}")
    private String address;

```

```
    @Value("${xxl.job.executor.ip}")
    private String ip;

    @Value("${xxl.job.executor.port}")
    private int port;

    @Value("${xxl.job.executor.logpath}")
    private String logPath;

    @Value("${xxl.job.executor.logretentiondays}")
    private int logRetentionDays;

    @Bean
    public XxlJobSpringExecutor xxlJobExecutor() {
        logger.info(">>>>>>>>>>> xxl-job config init.");
        XxlJobSpringExecutor xxlJobSpringExecutor = new XxlJobSpringExecutor();
        xxlJobSpringExecutor.setAdminAddresses(adminAddresses);
        xxlJobSpringExecutor.setAppName(appname);
        xxlJobSpringExecutor.setIp(ip);
        xxlJobSpringExecutor.setPort(port);
        xxlJobSpringExecutor.setAccessToken(accessToken);
        xxlJobSpringExecutor.setLogPath(logPath);
        xxlJobSpringExecutor.setLogRetentionDays(logRetentionDays);

        return xxlJobSpringExecutor;
    }


}
```

此时启动项目后，执行器就可用被识别了。

Java

```java
@Component
public class DemoJobHandler{

    @XxlJob("demoJobHandler")
    public ReturnT<String> demoJobHandler(String param) throws Exception {
        XxlJobLogger.log("XXL-JOB, Hello World.");
        System.out.println("测试执行器!");
        return ReturnT.SUCCESS;
    }
}
```

日志：logback.xml

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false" scan="true" scanPeriod="1 seconds">

    <contextName>logback</contextName>
    <property name="log.path" value="/data/applogs/xxl-job/xxl-job-executor-sample-springboot.log"/>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.path}</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${log.path}.%d{yyyy-MM-dd}.zip</fileNamePattern>
        </rollingPolicy>
        <encoder>
            <pattern>%date %level [%thread] %logger{36} [%file : %line] %msg%n
            </pattern>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="console"/>
        <appender-ref ref="file"/>
    </root>

</configuration>
```

yml

```yaml
logging:
  config: classpath:logback.xml
```