# 一、什么是xxl-job

xxl-job是一个分布式任务调度平台，其核心设计目标是开发迅速、学习简单、轻量级、易扩展。现已开放源代码并接入多家公司线上产品线，开箱即用。

官网：https://www.xuxueli.com/xxl-job/

## 1、**特性**

**简单**：支持通过Web页面对任务进行CRUD操作，操作简单，一分钟上手；

**动态**：支持动态修改任务状态、启动/停止任务，以及终止运行中任务，即时生效；

**调度中心HA（中心式）**：调度采用中心式设计，“调度中心”自研调度组件并支持集群部署，可保证调度中心HA；

**执行器HA（分布式）**：任务分布式执行，任务”执行器”支持集群部署，可保证任务执行HA；

**注册中心**: 执行器会周期性自动注册任务, 调度中心将会自动发现注册的任务并触发执行。同时，也支持手动录入执行器地址；

**弹性扩容缩容**：一旦有新执行器机器上线或者下线，下次调度时将会重新分配任务；

**触发策略**：提供丰富的任务触发策略，包括：Cron触发、固定间隔触发、固定延时触发、API（事件）触发、人工触发、父子任务触发；

**调度过期策略**：调度中心错过调度时间的补偿处理策略，包括：忽略、立即补偿触发一次等；

**阻塞处理策略**：调度过于密集执行器来不及处理时的处理策略，策略包括：单机串行（默认）、丢弃后续调度、覆盖之前调度；

**任务超时控制**：支持自定义任务超时时间，任务运行超时将会主动中断任务；

**任务失败重试**：支持自定义任务失败重试次数，当任务失败时将会按照预设的失败重试次数主动进行重试；其中分片任务支持分片粒度的失败重试；

任务失败告警；默认提供邮件方式失败告警，同时预留扩展接口，可方便的扩展短信、钉钉等告警方式；

**路由策略**：执行器集群部署时提供丰富的路由策略，包括：第一个、最后一个、轮询、随机、一致性HASH、最不经常使用、最近最久未使用、故障转移、忙碌转移等；

**分片广播任务**：执行器集群部署时，任务路由策略选择”分片广播”情况下，一次任务调度将会广播触发集群中所有执行器执行一次任务，可根据分片参数开发分片任务；

**动态分片**：分片广播任务以执行器为维度进行分片，支持动态扩容执行器集群从而动态增加分片数量，协同进行业务处理；在进行大数据量业务操作时可显著提升任务处理能力和速度。

**故障转移**：任务路由策略选择”故障转移”情况下，如果执行器集群中某一台机器故障，将会自动Failover切换到一台正常的执行器发送调度请求。

**任务进度监控**：支持实时监控任务进度；

**Rolling实时日志**：支持在线查看调度结果，并且支持以Rolling方式实时查看执行器输出的完整的执行日志；

**GLUE**：提供Web IDE，支持在线开发任务逻辑代码，动态发布，实时编译生效，省略部署上线的过程。支持30个版本的历史版本回溯。

**脚本任务**：支持以GLUE模式开发和运行脚本任务，包括Shell、Python、NodeJS、PHP、PowerShell等类型脚本;

**命令行任务**：原生提供通用命令行任务Handler（Bean任务，”CommandJobHandler”）；业务方只需要提供命令行即可；

**任务依赖**：支持配置子任务依赖，当父任务执行结束且执行成功后将会主动触发一次子任务的执行, 多个子任务用逗号分隔；

**一致性**：“调度中心”通过DB锁保证集群分布式调度的一致性, 一次任务调度只会触发一次执行；

**自定义任务参数**：支持在线配置调度任务入参，即时生效；

**调度线程池**：调度系统多线程触发调度运行，确保调度精确执行，不被堵塞；

**数据加密**：调度中心和执行器之间的通讯进行数据加密，提升调度信息安全性；

**邮件报警**：任务失败时支持邮件报警，支持配置多邮件地址群发报警邮件；

**推送maven中央仓库**: 将会把最新稳定版推送到maven中央仓库, 方便用户接入和使用;

**运行报表**：支持实时查看运行数据，如任务数量、调度次数、执行器数量等；以及调度报表，如调度日期分布图，调度成功分布图等；

**全异步**：任务调度流程全异步化设计实现，如异步调度、异步运行、异步回调等，有效对密集调度进行流量削峰，理论上支持任意时长任务的运行；

**跨语言**：调度中心与执行器提供语言无关的 RESTful API 服务，第三方任意语言可据此对接调度中心或者实现执行器。除此之外，还提供了 “多任务模式”和“httpJobHandler”等其他跨语言方案；

**国际化**：调度中心支持国际化设置，提供中文、英文两种可选语言，默认为中文；

**容器化**：提供官方docker镜像，并实时更新推送dockerhub，进一步实现产品开箱即用；

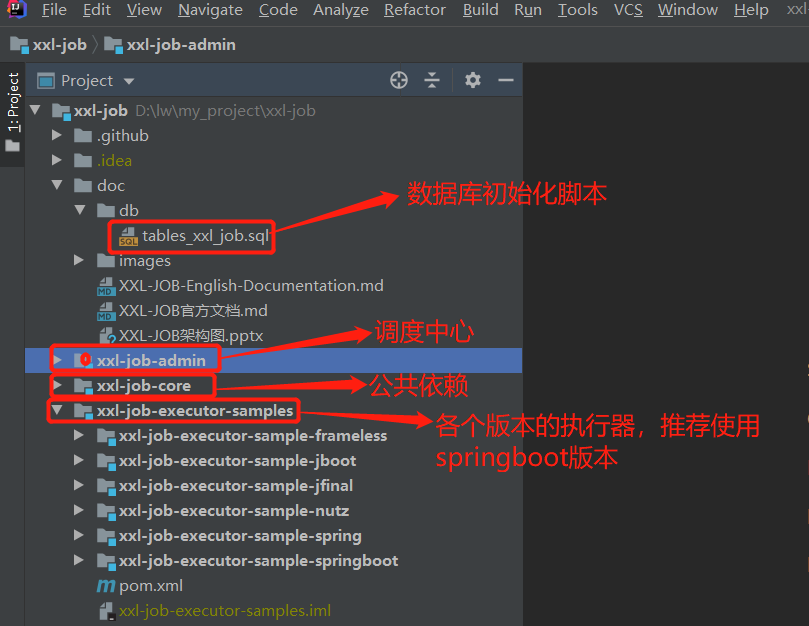
**线程池隔离**：调度线程池进行隔离拆分，慢任务自动降级进入”Slow”线程池，避免耗尽调度线程，提高系统稳定性；

**用户管理**：支持在线管理系统用户，存在管理员、普通用户两种角色；

**权限控制**：执行器维度进行权限控制，管理员拥有全量权限，普通用户需要分配执行器权限后才允许相关操作；

# 二、快速入门

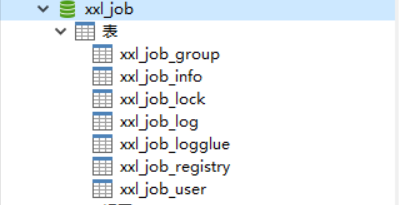
1.下载需要的版本https://github.com/xuxueli/xxl-job/



# ****初始化数据库脚本****

## 1、执行数据库脚本

 /doc/db/tables\_xxl\_job.sql



xxl\_job\_lock：任务调度锁表；

xxl\_job\_group：执行器信息表，维护任务执行器信息；

xxl\_job\_info：调度扩展信息表： 用于保存XXL-JOB调度任务的扩展信息，如任务分组、任务名、机器地址、执行器、执行入参和报警邮件等等；

xxl\_job\_log：调度日志表： 用于保存XXL-JOB任务调度的历史信息，如调度结果、执行结果、调度入参、调度机器和执行器等等；

xxl\_job\_log\_report：调度日志报表：用户存储XXL-JOB任务调度日志的报表，调度中心报表功能页面会用到；

xxl\_job\_logglue：任务GLUE日志：用于保存GLUE更新历史，用于支持GLUE的版本回溯功能；

xxl\_job\_registry：执行器注册表，维护在线的执行器和调度中心机器地址信息；

xxl\_job\_user：系统用户表；

# **配置调度中心xxl-job-admin**

****修改调度中心配置文件：/xxl-job/xxl-job-admin/src/main/resources/application.properties****

### 1、调度中心项目的端口号以及访问路径

### web

server.port=8080

server.context-path=/xxl-job-admin

### 2、配置静态文件的前缀

### resources

spring.mvc.static-path-pattern=/static/\*\*

spring.resources.static-locations=classpath:/static/

### 3、配置模板文件

### freemarker

spring.freemarker.templateLoaderPath=classpath:/templates/

spring.freemarker.suffix=.ftl

spring.freemarker.charset=UTF-8

spring.freemarker.request-context-attribute=request

spring.freemarker.settings.number\_format=0.##########

### 4、配置mybatis的mapper文件地址

### mybatis

mybatis.mapper-locations=classpath:/mybatis-mapper/\*Mapper.xml

### 5、配置数据库的地址

### xxl-job, datasource

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/xxl\_job?Unicode=true&characterEncoding=UTF-8

spring.datasource.username=root

spring.datasource.password=123456

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource

spring.datasource.tomcat.max-wait=10000

spring.datasource.tomcat.max-active=30

spring.datasource.tomcat.test-on-borrow=true

spring.datasource.tomcat.validation-query=SELECT 1

spring.datasource.tomcat.validation-interval=30000

### 6、配置报警邮箱

### xxl-job email

spring.mail.host=smtp.qq.com

spring.mail.port=25

spring.mail.username=xxx @qq.com

spring.mail.password=xxx

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.mail.properties.mail.smtp.starttls.required=true

spring.mail.properties.mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory

### 7、管理界面的登录用户名密码

xxl.job.login.username=admin

xxl.job.login.password=123456

### 8、调度中心通讯TOKEN，非空时启用

### xxl-job, access token

xxl.job.accessToken=

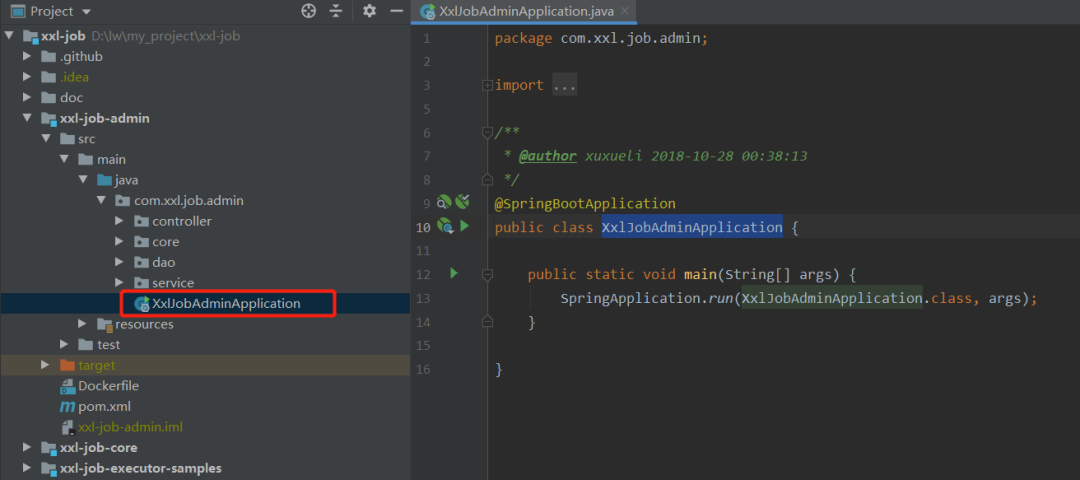
### 9、调度中心国际化设置，默认为中文版本，值设置为“en”时切换为英文版本

### xxl-job, i18n (default empty as chinese, "en" as english)

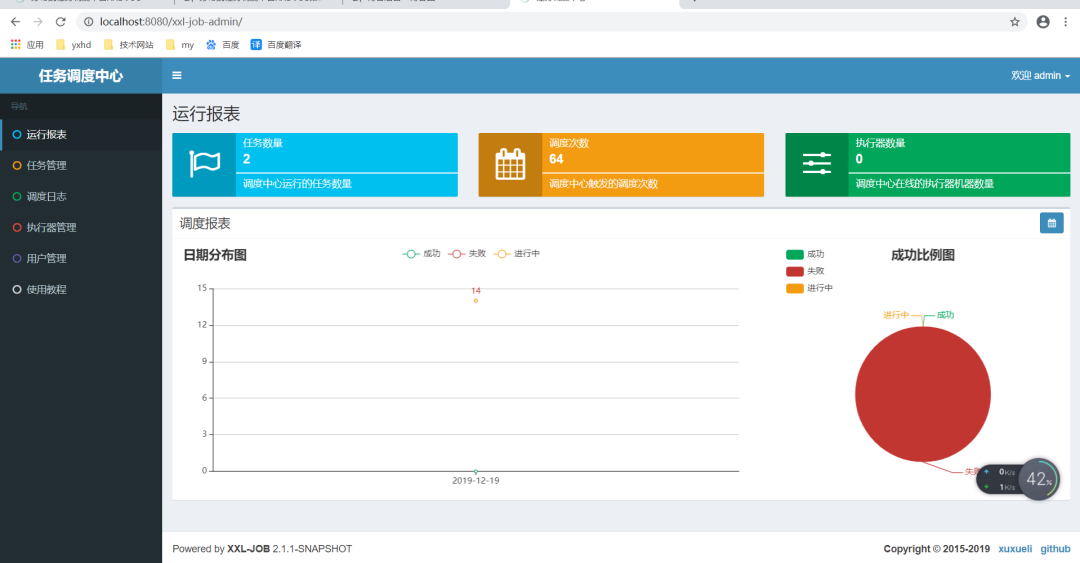
xxl.job.i18n=

# ****启动调度中心****

## 执行启动类XxlJobAdminApplication

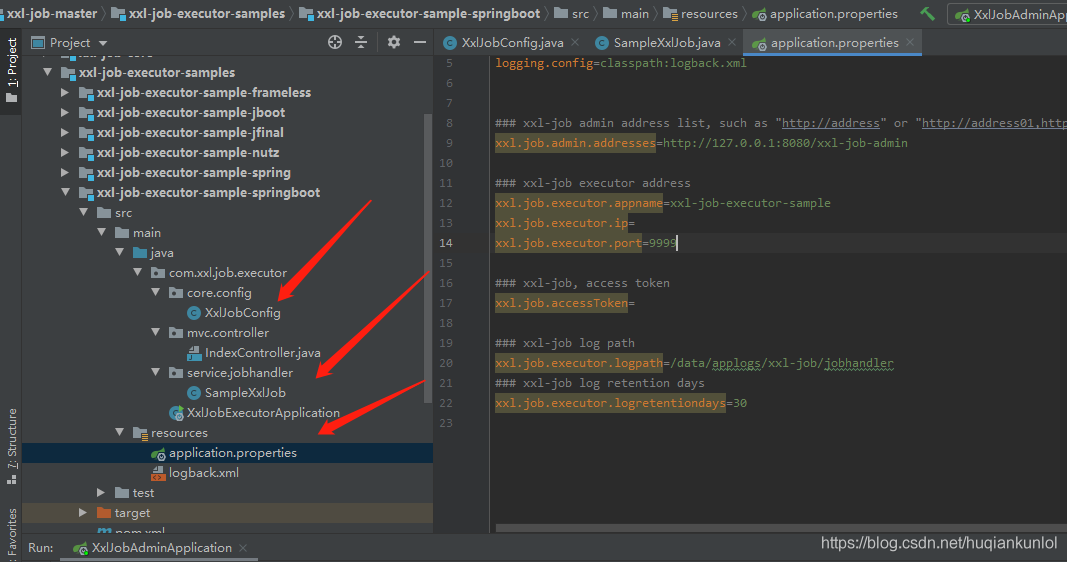


## **访问调度中心管理界面**



# ****六、创建执行器项目****

## 以项目中的xxl-job-executor-sample-springboot模块为例



## maven依赖

<!-- xxl-job-core -->

<dependency>

　　<groupId>com.xuxueli</groupId>

   <artifactId>xxl-job-core</artifactId>

   <version>${project.parent.version}</version>

</dependency>

## ****配置文件****

#项目端口号

# web port

server.port=8081

#日志文件

# log config

logging.config=classpath:logback.xml

#调度中心部署跟地址：如调度中心集群部署存在多个地址则用逗号分隔。

#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"。

### xxl-job admin address list, such as "http://address" or "http://address01,http://address02"

xxl.job.admin.addresses=http://127.0.0.1:8080/xxl-job-admin

#执行器的名称、ip地址、端口号

### xxl-job executor address

xxl.job.executor.appname=xxl-job-executor-sample

xxl.job.executor.ip=

xxl.job.executor.port=9999

#执行器通讯TOKEN：非空时启用

### xxl-job, access token

xxl.job.accessToken=

#执行器运行日志文件存储的磁盘位置，需要对该路径拥有读写权限

### xxl-job log path

xxl.job.executor.logpath=/data/applogs/xxl-job/jobhandler

#执行器Log文件定期清理功能，指定日志保存天数，日志文件过期自动删除。限制至少保持3天，否则功能不生效

#-1表示永不删除

### xxl-job log retention days

xxl.job.executor.logretentiondays=-1

注意：配置执行器的名称、IP地址、端口号，后面如果配置多个执行器时，要****防止端口冲突****

## ****配置类XxlJobConfig.java****

package com.xxl.job.executor.core.config;

import com.xxl.job.core.executor.impl.XxlJobSpringExecutor;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class XxlJobConfig {

    private Logger logger = LoggerFactory.getLogger(XxlJobConfig.class);

    @Value("${xxl.job.admin.addresses}")

    private String adminAddresses;

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

    private String appName;

    @Value("${xxl.job.executor.ip}")

    private String ip;

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

    private int port;

    @Value("${xxl.job.accessToken}")

    private String accessToken;

    @Value("${xxl.job.executor.logpath}")

    private String logPath;

    @Value("${xxl.job.executor.logretentiondays}")

    private int logRetentionDays;

    @Bean(initMethod = "start", destroyMethod = "destroy")

    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;

    }

    /\*\*

     \* 针对多网卡、容器内部署等情况，可借助 "spring-cloud-commons" 提供的 "InetUtils" 组件灵活定制注册IP；

     \*

     \*      1、引入依赖：

     \*          <dependency>

     \*             <groupId>org.springframework.cloud</groupId>

     \*             <artifactId>spring-cloud-commons</artifactId>

     \*             <version>${version}</version>

     \*         </dependency>

     \*

     \*      2、配置文件，或者容器启动变量

     \*          spring.cloud.inetutils.preferred-networks: 'xxx.xxx.xxx.'

     \*

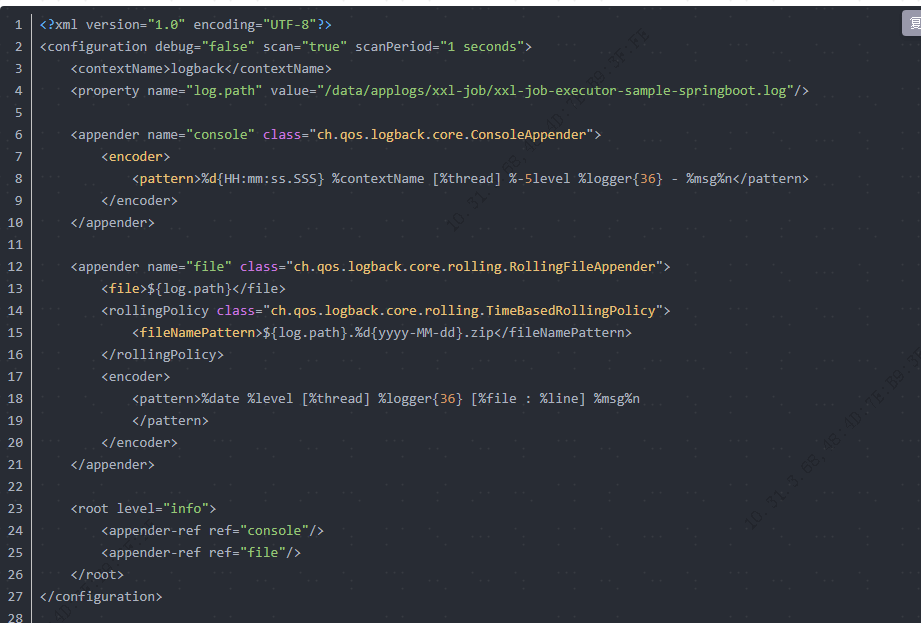
     \*      3、获取IP

     \*          String ip\_ = inetUtils.findFirstNonLoopbackHostInfo().getIpAddress();

     \*/

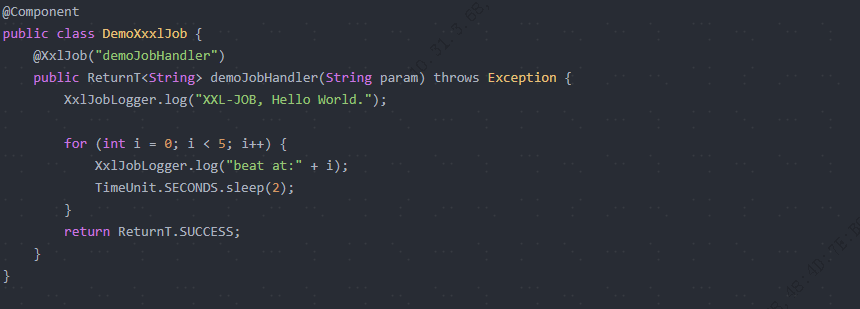
}

## 添加 logback.xml

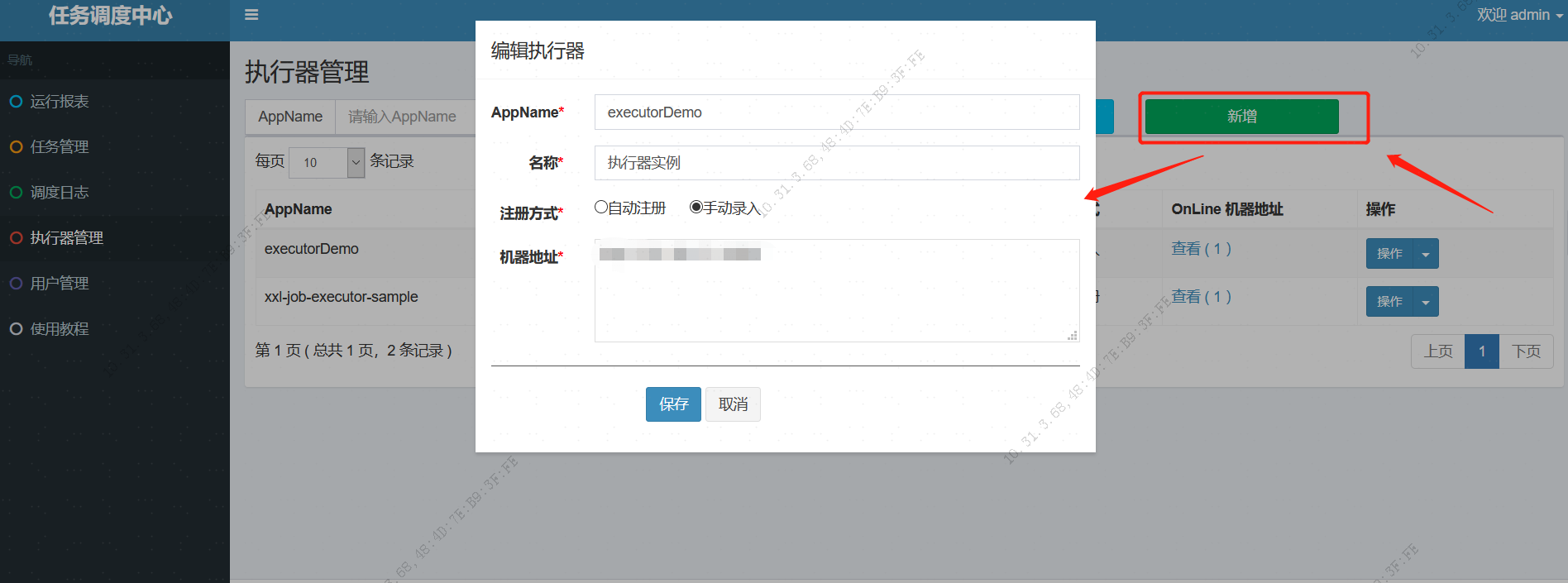


## ****5、创建任务JobHandler****

### 1、新建执行器



## 6、在调度中心中配置执行器



****参数介绍：****

****AppName****：是每个执行器集群的唯一标示AppName, 执行器会周期性以AppName为对象进行自动注册。可通过该配置自动发现注册成功的执行器, 供任务调度时使用;

****名称****：执行器的名称, 因为AppName限制字母数字等组成,可读性不强, 名称为了提高执行器的可读性;

****排序****: 执行器的排序, 系统中需要执行器的地方,如任务新增, 将会按照该排序读取可用的执行器列表;

****注册方式****：调度中心获取执行器地址的方式，

****自动注册****：执行器自动进行执行器注册，调度中心通过底层注册表可以动态发现执行器机器地址；

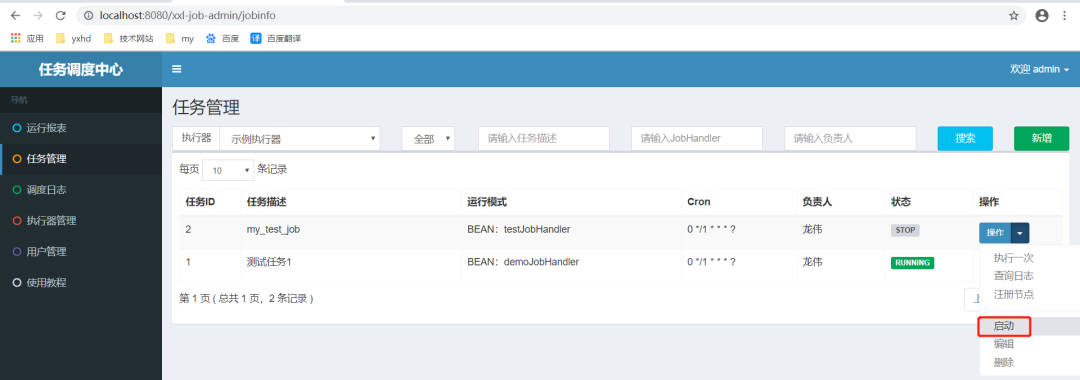
****手动录入****：人工手动录入执行器的地址信息，多地址逗号分隔，供调度中心使用；

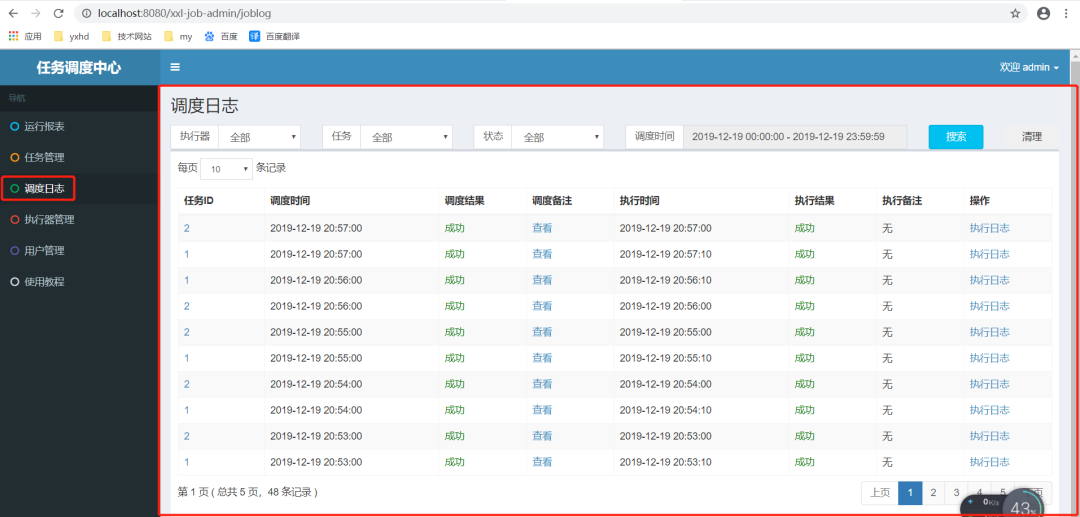
****机器地址****："注册方式"为"手动录入"时有效，支持人工维护执行器的地址信息；

## 7、创建任务，点击 任务管理--->新增任务--->如下界面，然后填充此表格，点击保存



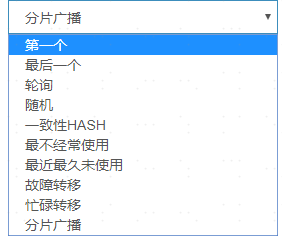
## 启动任务及查看日志





# **七、分片广播（集群模式）**

## 1、路由策略



## 2、分片任务执行器

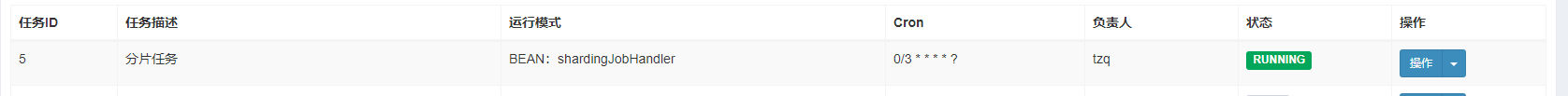


## 3、添加任务



1. 开启任务





1. 查看日志

