[**一、简介**](http://www.xuxueli.com/xxl-job/#/?id=%e4%b8%80%e3%80%81%e7%ae%80%e4%bb%8b)

[**1.1 概述**](http://www.xuxueli.com/xxl-job/#/?id=_11-%e6%a6%82%e8%bf%b0)

XXL-JOB是一个轻量级分布式任务调度框架，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。

[**1.2 特性**](http://www.xuxueli.com/xxl-job/#/?id=_12-%e7%89%b9%e6%80%a7)

* 1、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；
* 2、动态：支持动态修改任务状态、暂停/恢复任务，以及终止运行中任务，即时生效；
* 3、调度中心HA（中心式）：调度采用中心式设计，“调度中心”基于集群Quartz实现，可保证调度中心HA；
* 4、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；
* 5、任务Failover：执行器集群部署时，任务路由策略选择"故障转移"情况下调度失败时将会平滑切换执行器进行Failover；
* 6、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；
* 7、自定义任务参数：支持在线配置调度任务入参，即时生效；
* 8、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；
* 9、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；
* 10、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；
* 11、状态监控：支持实时监控任务进度；
* 12、Rolling执行日志：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；
* 13、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。
* 14、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；
* 15、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；
* 16、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;
* 17、任务注册: 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；
* 18、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；
* 19、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；
* 20、脚本任务：支持以GLUE模式开发和运行脚本任务，包括Shell、Python等类型脚本;
* 21、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；
* 22、失败处理策略；调度失败时的处理策略，策略包括：失败告警（默认）、失败重试；
* 23、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；
* 24、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。
* 25、事件触发：除了"Cron方式"和"任务依赖方式"触发任务执行之外，支持基于事件的触发任务方式。调度中心提供触发任务单次执行的API服务，可根据业务事件灵活触发。

[**1.3 发展**](http://www.xuxueli.com/xxl-job/#/?id=_13-%e5%8f%91%e5%b1%95)

于2015年中，我在github上创建XXL-JOB项目仓库并提交第一个commit，随之进行系统结构设计，UI选型，交互设计……

于2015-11月，XXL-JOB终于RELEASE了第一个大版本V1.0， 随后我将之发布到OSCHINA，XXL-JOB在OSCHINA上获得了@红薯的热门推荐，同期分别达到了OSCHINA的“热门动弹”排行第一和git.oschina的开源软件月热度排行第一，在此特别感谢红薯，感谢大家的关注和支持。

于2015-12月，我将XXL-JOB发表到我司内部知识库，并且得到内部同事认可。

于2016-01月，我司展开XXL-JOB的内部接入和定制工作，在此感谢袁某和尹某两位同事的贡献，同时也感谢内部其他给与关注与支持的同事。

于2017-05-13，在上海举办的 "[**第62期开源中国源创会**](https://www.oschina.net/event/2236961)" 的 "放码过来" 环节，我登台对XXL-JOB做了演讲，台下五百位在场观众反响热烈（[**图文回顾**](https://www.oschina.net/question/2686220_2242120) ）。

**我司大众点评目前已接入XXL-JOB，内部别名《Ferrari》（Ferrari基于XXL-JOB的V1.1版本定制而成，新接入应用推荐升级最新版本）。 据最新统计, 自2016-01-21接入至2017-07-07期间，该系统已调度约60万余次，表现优异。新接入应用推荐使用最新版本，因为经过数个大版本的更新，系统的任务模型、UI交互模型以及底层调度通讯模型都有了较大的优化和提升，核心功能更加稳定高效。**

至今，XXL-JOB已接入多家公司的线上产品线，接入场景如电商业务，O2O业务和大数据作业等，截止2016-07-19为止，XXL-JOB已接入的公司包括不限于：

- 1、大众点评；

- 2、山东学而网络科技有限公司；

- 3、安徽慧通互联科技有限公司；

- 4、人人聚财金服；

- 5、上海棠棣信息科技股份有限公司

- 6、运满满

- 7、米其林 (中国区)

- 8、妈妈联盟

- 9、九樱天下（北京）信息技术有限公司

- 10、万普拉斯科技有限公司(一加手机)

- 11、上海亿保健康管理有限公司

- 12、海尔馨厨 (海尔)

- 13、河南大红包电子商务有限公司

- 14、成都顺点科技有限公司

- 15、深圳市怡亚通

- 16、深圳麦亚信科技股份有限公司

- 17、上海博莹科技信息技术有限公司

- 18、中国平安科技有限公司

- 19、杭州知时信息科技有限公司

- 20、博莹科技（上海）有限公司

- 21、成都依能股份有限责任公司

- 22、湖南高阳通联信息技术有限公司

- 23、深圳市邦德文化发展有限公司

- 24、福建阿思可网络教育有限公司

- 25、优信二手车

- 26、上海悠游堂投资发展股份有限公司

- 27、北京粉笔蓝天科技有限公司

- 28、中秀科技(无锡)有限公司

- 29、武汉空心科技有限公司

- 30、北京蚂蚁风暴科技有限公司

- 31、四川互宜达科技有限公司

- 32、钱包行云（北京）科技有限公司

- 33、重庆欣才集团

- 34、咪咕互动娱乐有限公司（中国移动）

- 35、北京诺亦腾科技有限公司

- 36、增长引擎(北京)信息技术有限公司

- 37、北京英贝思科技有限公司

- 38、刚泰集团

- 39、深圳泰久信息系统股份有限公司

- 40、随行付支付有限公司

- 41、广州瀚农网络科技有限公司

- 42、享点科技有限公司

- 43、杭州比智科技有限公司

- 44、圳临界线网络科技有限公司

- 45、广州知识圈网络科技有限公司

- 46、国誉商业上海有限公司

- 47、海尔消费金融有限公司，嗨付、够花 (海尔)

- ……

**更多接入的公司，欢迎在**[**登记地址**](https://github.com/xuxueli/xxl-job/issues/1)**登记，登记仅仅为了产品推广。**

欢迎大家的关注和使用，XXL-JOB也将拥抱变化，持续发展。

[**1.4 下载**](http://www.xuxueli.com/xxl-job/#/?id=_14-%e4%b8%8b%e8%bd%bd)

[**文档地址**](http://www.xuxueli.com/xxl-job/#/?id=%e6%96%87%e6%a1%a3%e5%9c%b0%e5%9d%80)

* [**中文文档**](http://www.xuxueli.com/xxl-job/)
* [**English Documentation**](http://www.xuxueli.com/xxl-job/en/)

[**源码仓库地址**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ba%90%e7%a0%81%e4%bb%93%e5%ba%93%e5%9c%b0%e5%9d%80)

| **源码仓库地址** | **Release Download** |
| --- | --- |
| [**https://github.com/xuxueli/xxl-job**](https://github.com/xuxueli/xxl-job) | [**Download**](https://github.com/xuxueli/xxl-job/releases) |
| [**http://git.oschina.net/xuxueli0323/xxl-job**](http://git.oschina.net/xuxueli0323/xxl-job) | [**Download**](http://git.oschina.net/xuxueli0323/xxl-job/releases) |

[**中央仓库地址**](http://www.xuxueli.com/xxl-job/#/?id=%e4%b8%ad%e5%a4%ae%e4%bb%93%e5%ba%93%e5%9c%b0%e5%9d%80)

<!-- http://repo1.maven.org/maven2/com/xuxueli/xxl-job-core/ -->

<dependency>

<groupId>com.xuxueli</groupId>

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

<version>1.8.2</version>

</dependency>

[**技术交流**](http://www.xuxueli.com/xxl-job/#/?id=%e6%8a%80%e6%9c%af%e4%ba%a4%e6%b5%81)

* [**社区交流**](http://www.xuxueli.com/page/community.html)
* [**Gitter**](https://gitter.im/xuxueli/xxl-job)

[**1.5 环境**](http://www.xuxueli.com/xxl-job/#/?id=_15-%e7%8e%af%e5%a2%83)

* JDK：1.7+
* Servlet/JSP Spec：3.1/2.3
* Tomcat：8.5.x/Jetty9.2.x
* Spring-boot：1.5.x/Spring4.x
* Mysql：5.6+
* Maven：3+

[**二、快速入门**](http://www.xuxueli.com/xxl-job/#/?id=%e4%ba%8c%e3%80%81%e5%bf%ab%e9%80%9f%e5%85%a5%e9%97%a8)

[**2.1 初始化“调度数据库”**](http://www.xuxueli.com/xxl-job/#/?id=_21-%e5%88%9d%e5%a7%8b%e5%8c%96%e8%b0%83%e5%ba%a6%e6%95%b0%e6%8d%ae%e5%ba%93)

请下载项目源码并解压，获取 "调度数据库初始化SQL脚本" 并执行即可，正常情况下应该生成16张表。

"调度数据库初始化SQL脚本" 位置为:

/xxl-job/doc/db/tables\_xxl\_job.sql

调度中心支持集群部署，集群情况下各节点务必连接同一个mysql实例;

如果mysql做主从,调度中心集群节点务必强制走主库;

[**2.2 编译源码**](http://www.xuxueli.com/xxl-job/#/?id=_22-%e7%bc%96%e8%af%91%e6%ba%90%e7%a0%81)

解压源码,按照maven格式将源码导入IDE, 使用maven进行编译即可，源码结构如下：

xxl-job-admin：调度中心

xxl-job-core：公共依赖

xxl-job-executor：执行器Sample示例（选择合适的版本执行器，可直接使用，也可以参考其并将现有项目改造成执行器）

：xxl-job-executor-sample-spring：Spring版本，通过Spring容器管理执行器，比较通用，推荐这种方式；

：xxl-job-executor-sample-springboot：Springboot版本，通过Springboot管理执行器；

：xxl-job-executor-sample-jfinal：JFinal版本，通过JFinal管理执行器；

[**2.3 配置部署“调度中心”**](http://www.xuxueli.com/xxl-job/#/?id=_23-%e9%85%8d%e7%bd%ae%e9%83%a8%e7%bd%b2%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83)

调度中心项目：xxl-job-admin

作用：统一管理任务调度平台上调度任务，负责触发调度执行，并且提供任务管理平台。

[**步骤一：调度中心配置：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%e9%85%8d%e7%bd%ae%ef%bc%9a)

调度中心配置文件地址：

/xxl-job/xxl-job-admin/src/main/resources/xxl-job-admin.properties

调度中心配置内容说明：

### 调度中心JDBC链接：链接地址请保持和 2.1章节 所创建的调度数据库的地址一致

xxl.job.db.driverClass=com.mysql.jdbc.Driver

xxl.job.db.url=jdbc:mysql://localhost:3306/xxl-job?useUnicode=true&characterEncoding=UTF-8

xxl.job.db.user=root

xxl.job.db.password=root\_pwd

### 报警邮箱

xxl.job.mail.host=smtp.163.com

xxl.job.mail.port=25

xxl.job.mail.username=ovono802302@163.com

xxl.job.mail.password=asdfzxcv

xxl.job.mail.sendFrom=ovono802302@163.com

xxl.job.mail.sendNick=《任务调度平台XXL-JOB》

### 登录账号

xxl.job.login.username=admin

xxl.job.login.password=123456

### 调度中心通讯TOKEN，非空时启用

xxl.job.accessToken=

[**步骤二：部署项目：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e9%83%a8%e7%bd%b2%e9%a1%b9%e7%9b%ae%ef%bc%9a)

如果已经正确进行上述配置，可将项目编译打war包并部署到tomcat中。 调度中心访问地址：[**http://localhost:8080/xxl-job-admin**](http://localhost:8080/xxl-job-admin) (该地址执行器将会使用到，作为回调地址)，登录后运行界面如下图所示



至此“调度中心”项目已经部署成功。

[**步骤三：调度中心集群（可选）：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%89%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%e9%9b%86%e7%be%a4%ef%bc%88%e5%8f%af%e9%80%89%ef%bc%89%ef%bc%9a)

调度中心支持集群部署，提升调度系统可用性。

集群部署唯一要求为：保证每个集群节点配置（db和登陆账号等）保持一致。调度中心通过db配置区分不同集群。

调度中心在集群部署时可通过nginx负载均衡，此时可以为集群分配一个域名。该域名一方面可以用于访问，另一方面也可以用于配置执行器回调地址。

[**2.4 配置部署“执行器项目”**](http://www.xuxueli.com/xxl-job/#/?id=_24-%e9%85%8d%e7%bd%ae%e9%83%a8%e7%bd%b2%e6%89%a7%e8%a1%8c%e5%99%a8%e9%a1%b9%e7%9b%ae)

“执行器”项目：xxl-job-executor-sample-spring (如新建执行器项目，可参考该Sample示例执行器项目的配置步骤；)

作用：负责接收“调度中心”的调度并执行；可直接部署执行器，也可以将执行器集成到现有业务项目中。

[**步骤一：maven依赖**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9amaven%e4%be%9d%e8%b5%96)

确认pom文件中引入了 "xxl-job-core" 的maven依赖；

[**步骤二：执行器配置**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e6%89%a7%e8%a1%8c%e5%99%a8%e9%85%8d%e7%bd%ae)

执行器配置，配置文件地址：

/xxl-job/xxl-job-executor-samples/xxl-job-executor-sample-spring/src/main/resources/xxl-job-executor.properties

执行器配置，配置内容说明：

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

### xxl-job executor address：执行器"AppName"和地址信息配置：AppName执行器心跳注册分组依据；地址信息用于"调度中心请求并触发任务"和"执行器注册"。执行器默认端口为9999，执行器IP默认为空表示自动获取IP，多网卡时可手动设置指定IP，手动设置IP时将会绑定Host。单机部署多个执行器时，注意要配置不同执行器端口；

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

xxl.job.executor.ip=

xxl.job.executor.port=9999

### xxl-job log path：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.executor.logpath=/data/applogs/xxl-job/jobhandler/

### xxl-job, access token：执行器通讯TOKEN，非空时启用

xxl.job.accessToken=

[**步骤三：执行器组件配置**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%89%ef%bc%9a%e6%89%a7%e8%a1%8c%e5%99%a8%e7%bb%84%e4%bb%b6%e9%85%8d%e7%bd%ae)

执行器组件，配置文件地址：

/xxl-job/xxl-job-executor-samples/xxl-job-executor-sample-spring/src/main/resources/applicationcontext-xxl-job.xml

执行器组件，配置内容说明：

<!-- 配置01、JobHandler 扫描路径：自动扫描容器中JobHandler -->

<context:component-scan base-package="com.xxl.job.executor.service.jobhandler" />

<!-- 配置02、执行器Excutor配置：执行器核心配置 -->

<bean id="xxlJobExecutor" class="com.xxl.job.core.executor.XxlJobExecutor" init-method="start" destroy-method="destroy" >

<!-- 执行器IP[选填]，为空则自动获取 -->

<property name="ip" value="${xxl.job.executor.ip}" />

<!-- 执行器端口号[必须] -->

<property name="port" value="${xxl.job.executor.port}" />

<!-- 执行器AppName[选填]，为空则关闭自动注册 -->

<property name="appName" value="${xxl.job.executor.appname}" />

<!-- 执行器注册中心地址[选填]，为空则关闭自动注册 -->

<property name="adminAddresses" value="${xxl.job.admin.addresses}" />

<!-- 执行器日志路径[必填] -->

<property name="logPath" value="${xxl.job.executor.logpath}" />

<!-- 访问令牌，非空则进行匹配校验[选填] -->

<property name="accessToken" value="${xxl.job.accessToken}" />

</bean>

[**步骤四：部署执行器项目：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e5%9b%9b%ef%bc%9a%e9%83%a8%e7%bd%b2%e6%89%a7%e8%a1%8c%e5%99%a8%e9%a1%b9%e7%9b%ae%ef%bc%9a)

如果已经正确进行上述配置，可将执行器项目编译打部署，系统提供三个执行器Sample示例项目，选择其中一个即可，各自的部署方式如下。

xxl-job-executor-sample-spring：项目编译打包成WAR包，并部署到tomcat中。

xxl-job-executor-sample-springboot：项目编译打包成springboot类型的可执行JAR包，命令启动即可；

至此“执行器”项目已经部署结束。

[**步骤五：执行器集群（可选）：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%94%ef%bc%9a%e6%89%a7%e8%a1%8c%e5%99%a8%e9%9b%86%e7%be%a4%ef%bc%88%e5%8f%af%e9%80%89%ef%bc%89%ef%bc%9a)

执行器支持集群部署，提升调度系统可用性，同时提升任务处理能力。

集群部署唯一要求为：保证集群中每个执行器的配置项 "xxl.job.admin.addresses/调度中心地址" 保持一致，执行器根据该配置进行执行器自动注册等操作。

[**2.5 开发第一个任务“Hello World”**](http://www.xuxueli.com/xxl-job/#/?id=_25-%e5%bc%80%e5%8f%91%e7%ac%ac%e4%b8%80%e4%b8%aa%e4%bb%bb%e5%8a%a1hello-world)

本示例以新建一个 “GLUE模式(Java)” 运行模式的任务为例。更多有关任务的详细配置，请查看“章节三：任务详解”。 （ “GLUE模式(Java)”的执行代码托管到调度中心在线维护，相比“Bean模式任务”需要在执行器项目开发部署上线，更加简便轻量）

**前提：请确认“调度中心”和“执行器”项目已经成功部署并启动；**

[**步骤一：新建任务：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e6%96%b0%e5%bb%ba%e4%bb%bb%e5%8a%a1%ef%bc%9a)

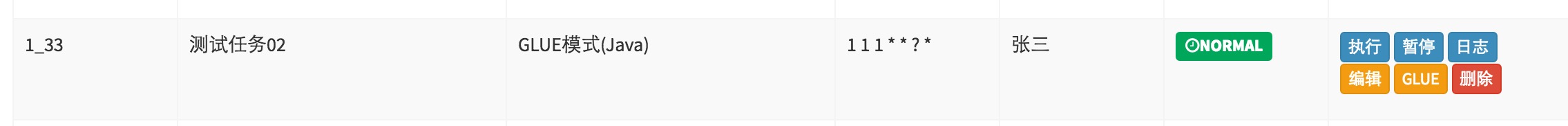
登录调度中心，点击下图所示“新建任务”按钮，新建示例任务。然后，参考下面截图中任务的参数配置，点击保存。

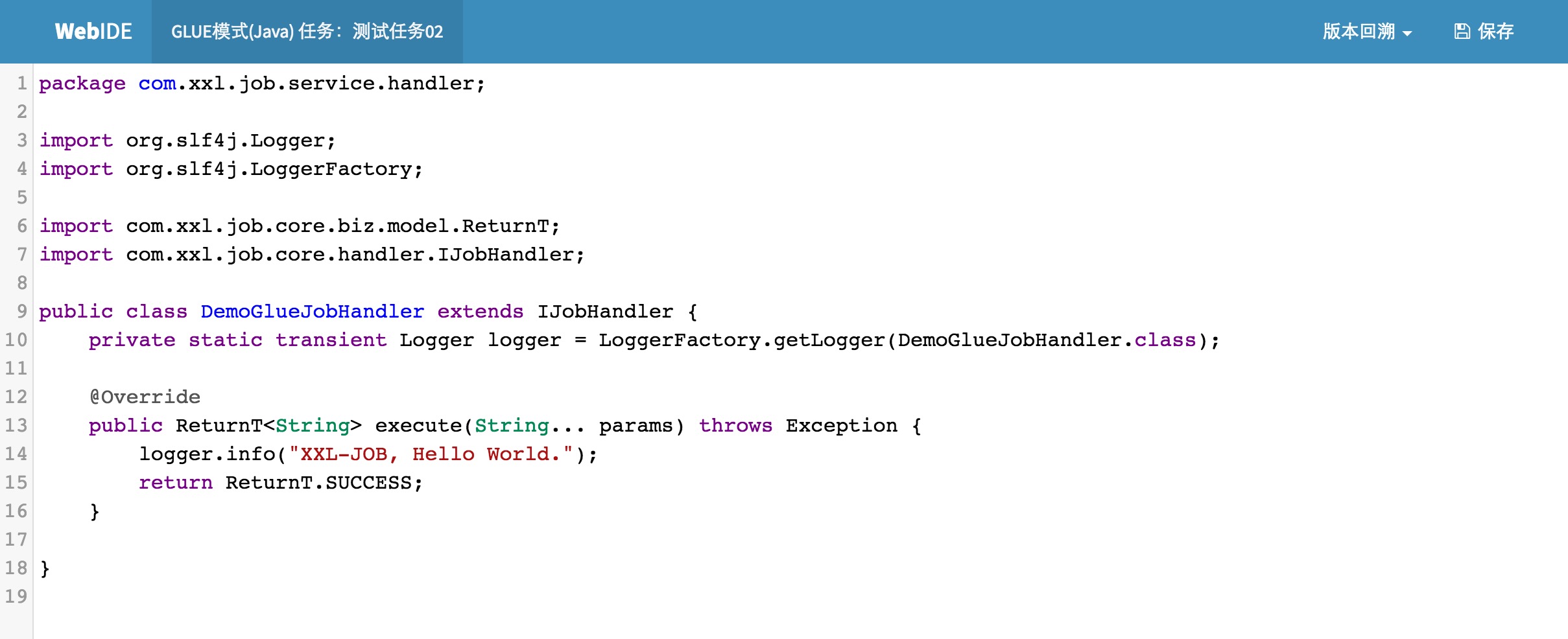




[**步骤二：“GLUE模式(Java)” 任务开发：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9aglue%e6%a8%a1%e5%bc%8fjava-%e4%bb%bb%e5%8a%a1%e5%bc%80%e5%8f%91%ef%bc%9a)

请点击任务右侧 “GLUE” 按钮，进入 “GLUE编辑器开发界面” ，见下图。“GLUE模式(Java)” 运行模式的任务默认已经初始化了示例任务代码，即打印Hello World。 （ “GLUE模式(Java)” 运行模式的任务实际上是一段继承自IJobHandler的Java类代码，它在执行器项目中运行，可使用@Resource/@Autowire注入执行器里中的其他服务，详细介绍请查看第三章节）





[**步骤三：触发执行：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%89%ef%bc%9a%e8%a7%a6%e5%8f%91%e6%89%a7%e8%a1%8c%ef%bc%9a)

请点击任务右侧 “执行” 按钮，可手动触发一次任务执行（通常情况下，通过配置Cron表达式进行任务调度出发）。

[**步骤四：查看日志：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e5%9b%9b%ef%bc%9a%e6%9f%a5%e7%9c%8b%e6%97%a5%e5%bf%97%ef%bc%9a)

请点击任务右侧 “日志” 按钮，可前往任务日志界面查看任务日志。 在任务日志界面中，可查看该任务的历史调度记录以及每一次调度的任务调度信息、执行参数和执行信息。运行中的任务点击右侧的“执行日志”按钮，可进入日志控制台查看实时执行日志。



在日志控制台，可以Rolling方式实时查看任务在执行器一侧运行输出的日志信息，实时监控任务进度；



[**三、任务详解**](http://www.xuxueli.com/xxl-job/#/?id=%e4%b8%89%e3%80%81%e4%bb%bb%e5%8a%a1%e8%af%a6%e8%a7%a3)

[**配置属性详细说明：**](http://www.xuxueli.com/xxl-job/#/?id=%e9%85%8d%e7%bd%ae%e5%b1%9e%e6%80%a7%e8%af%a6%e7%bb%86%e8%af%b4%e6%98%8e%ef%bc%9a)

- 执行器：任务的绑定的执行器，任务触发调度时将会自动发现注册成功的执行器, 实现任务自动发现功能; 另一方面也可以方便的进行任务分组。每个任务必须绑定一个执行器, 可在 "执行器管理" 进行设置;

- 描述：任务的描述信息，便于任务管理；

- 路由策略：当执行器集群部署时，提供丰富的路由策略，包括；

FIRST（第一个）：固定选择第一个执行器；

LAST（最后一个）：固定选择最后一个执行器；

ROUND（轮询）：；

RANDOM（随机）：随机选择在线的执行器；

CONSISTENT\_HASH（一致性HASH）：分组下机器地址相同，不同JOB均匀散列在不同机器上，保证分组下机器分配JOB平均；且每个JOB固定调度其中一台机器；

LEAST\_FREQUENTLY\_USED（最不经常使用）：单个JOB对应的每个执行器，使用频率最低的优先被选举；

LEAST\_RECENTLY\_USED（最近最久未使用）：单个JOB对应的每个执行器，最久为使用的优先被选举；

FAILOVER（故障转移）：按照顺序依次进行心跳检测，第一个心跳检测成功的机器选定为目标执行器并发起调度；

BUSYOVER（忙碌转移）：按照顺序依次进行空闲检测，第一个空闲检测成功的机器选定为目标执行器并发起调度；

SHARDING\_BROADCAST(分片广播)：广播触发对应集群中所有执行器执行一次任务，同时传递分片参数；可根据分片参数开发分片任务；

- Cron：触发任务执行的Cron表达式；

- 运行模式：

BEAN模式：任务以JobHandler方式维护在执行器端；需要结合 "JobHandler" 属性匹配执行器中任务；

GLUE模式(Java)：任务以源码方式维护在调度中心；该模式的任务实际上是一段继承自IJobHandler的Java类代码并 "groovy" 源码方式维护，它在执行器项目中运行，可使用@Resource/@Autowire注入执行器里中的其他服务；

GLUE模式(Shell)：任务以源码方式维护在调度中心；该模式的任务实际上是一段 "shell" 脚本；

GLUE模式(Python)：任务以源码方式维护在调度中心；该模式的任务实际上是一段 "python" 脚本；

GLUE模式(NodeJS)：任务以源码方式维护在调度中心；该模式的任务实际上是一段 "nodejs" 脚本；

- JobHandler：运行模式为 "BEAN模式" 时生效，对应执行器中新开发的JobHandler类“@JobHander”注解自定义的value值；

- 子任务Key：每个任务都拥有一个唯一的任务Key(任务Key可以从任务列表获取)，当本任务执行结束并且执行成功时，将会触发子任务Key所对应的任务的一次主动调度。

- 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；

单机串行（默认）：调度请求进入单机执行器后，调度请求进入FIFO队列并以串行方式运行；

丢弃后续调度：调度请求进入单机执行器后，发现执行器存在运行的调度任务，本次请求将会被丢弃并标记为失败；

覆盖之前调度：调度请求进入单机执行器后，发现执行器存在运行的调度任务，将会终止运行中的调度任务并清空队列，然后运行本地调度任务；

- 失败处理策略；调度失败时的处理策略；

失败告警（默认）：调度失败时，将会触发失败报警，如发送报警邮件；

失败重试：调度失败时，将会主动进行一次失败重试调度，重试调度后仍然失败将会触发一失败告警。注意当任务以failover方式路由时，每次失败重试将会触发新一轮路由。

- 执行参数：任务执行所需的参数，多个参数时用逗号分隔，任务执行时将会把多个参数转换成数组传入；

- 报警邮件：任务调度失败时邮件通知的邮箱地址，支持配置多邮箱地址，配置多个邮箱地址时用逗号分隔；

- 负责人：任务的负责人；

[**3.1 BEAN模式**](http://www.xuxueli.com/xxl-job/#/?id=_31-bean%e6%a8%a1%e5%bc%8f)

任务逻辑以JobHandler的形式存在于“执行器”所在项目中，开发流程如下：

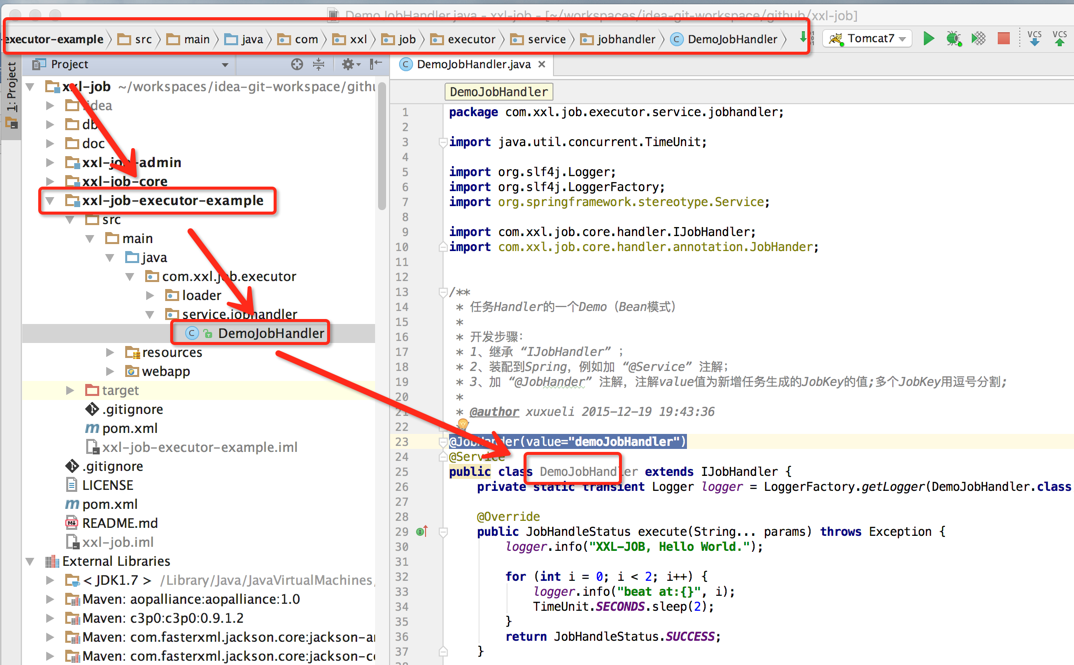
[**步骤一：执行器项目中，开发JobHandler：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e6%89%a7%e8%a1%8c%e5%99%a8%e9%a1%b9%e7%9b%ae%e4%b8%ad%ef%bc%8c%e5%bc%80%e5%8f%91jobhandler%ef%bc%9a)

- 1、 新建一个继承com.xxl.job.core.handler.IJobHandler的Java类；

- 2、 该类被Spring容器扫描为Bean实例，如加“@Component”注解；

- 3、 添加 “@JobHander(value="自定义jobhandler名称")”注解，注解的value值为自定义的JobHandler名称，该名称对应的是调度中心新建任务的JobHandler属性的值。

（可参考Sample示例执行器中的DemoJobHandler，见下图）



[**步骤二：调度中心，新建调度任务**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%ef%bc%8c%e6%96%b0%e5%bb%ba%e8%b0%83%e5%ba%a6%e4%bb%bb%e5%8a%a1)

参考上文“配置属性详细说明”对新建的任务进行参数配置，运行模式选中 "BEAN模式"，JobHandler属性填写任务注解@JobHander中定义的值；



[**3.2 GLUE模式(Java)**](http://www.xuxueli.com/xxl-job/#/?id=_32-glue%e6%a8%a1%e5%bc%8fjava)

任务以源码方式维护在调度中心，支持通过Web IDE在线更新，实时编译和生效，因此不需要指定JobHandler。开发流程如下：

[**步骤一：调度中心，新建调度任务：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%ef%bc%8c%e6%96%b0%e5%bb%ba%e8%b0%83%e5%ba%a6%e4%bb%bb%e5%8a%a1%ef%bc%9a)

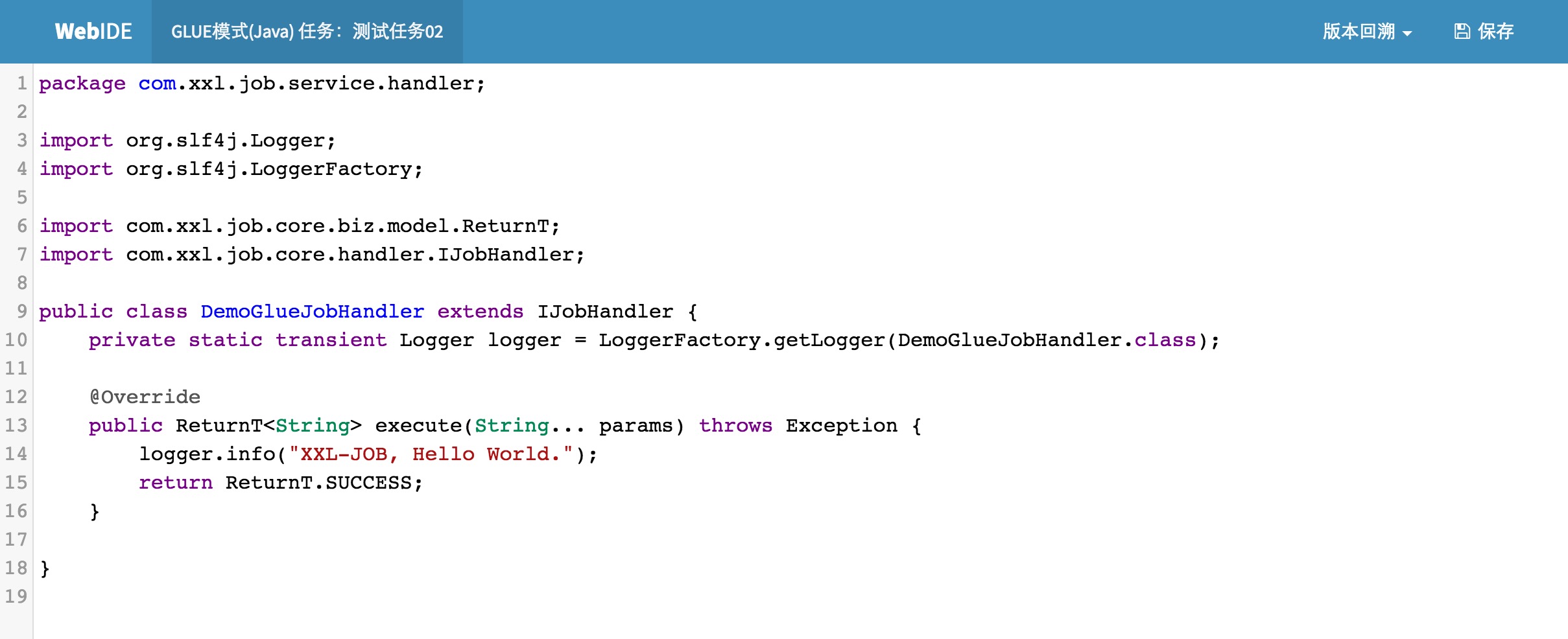
参考上文“配置属性详细说明”对新建的任务进行参数配置，运行模式选中 "GLUE模式(Java)"；



[**步骤二：开发任务代码：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e5%bc%80%e5%8f%91%e4%bb%bb%e5%8a%a1%e4%bb%a3%e7%a0%81%ef%bc%9a)

选中指定任务，点击该任务右侧“GLUE”按钮，将会前往GLUE任务的Web IDE界面，在该界面支持对任务代码进行开发（也可以在IDE中开发完成后，复制粘贴到编辑中）。

版本回溯功能（支持30个版本的版本回溯）：在GLUE任务的Web IDE界面，选择右上角下拉框“版本回溯”，会列出该GLUE的更新历史，选择相应版本即可显示该版本代码，保存后GLUE代码即回退到对应的历史版本；



[**3.3 GLUE模式(Shell)**](http://www.xuxueli.com/xxl-job/#/?id=_33-glue%e6%a8%a1%e5%bc%8fshell)

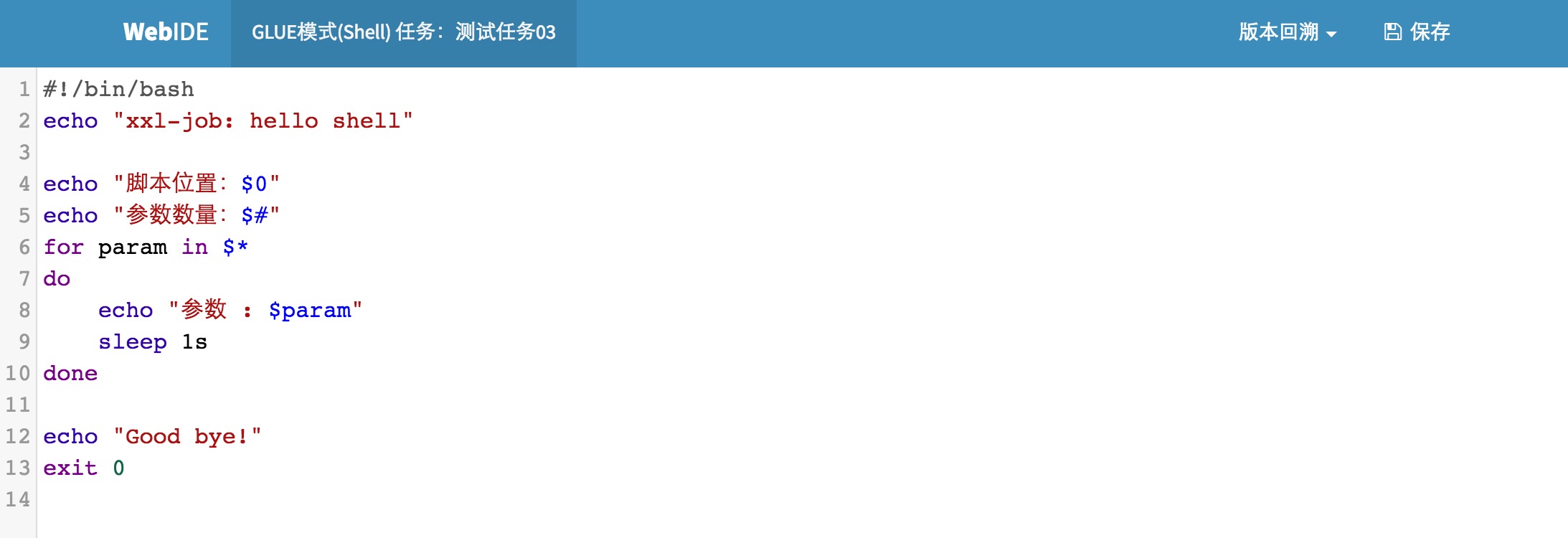
[**步骤一：调度中心，新建调度任务**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%ef%bc%8c%e6%96%b0%e5%bb%ba%e8%b0%83%e5%ba%a6%e4%bb%bb%e5%8a%a1)

参考上文“配置属性详细说明”对新建的任务进行参数配置，运行模式选中 "GLUE模式(Shell)"；

[**步骤二：开发任务代码：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e5%bc%80%e5%8f%91%e4%bb%bb%e5%8a%a1%e4%bb%a3%e7%a0%81%ef%bc%9a-1)

选中指定任务，点击该任务右侧“GLUE”按钮，将会前往GLUE任务的Web IDE界面，在该界面支持对任务代码进行开发（也可以在IDE中开发完成后，复制粘贴到编辑中）。

该模式的任务实际上是一段 "shell" 脚本；



[**3.4 GLUE模式(Python)**](http://www.xuxueli.com/xxl-job/#/?id=_34-glue%e6%a8%a1%e5%bc%8fpython)

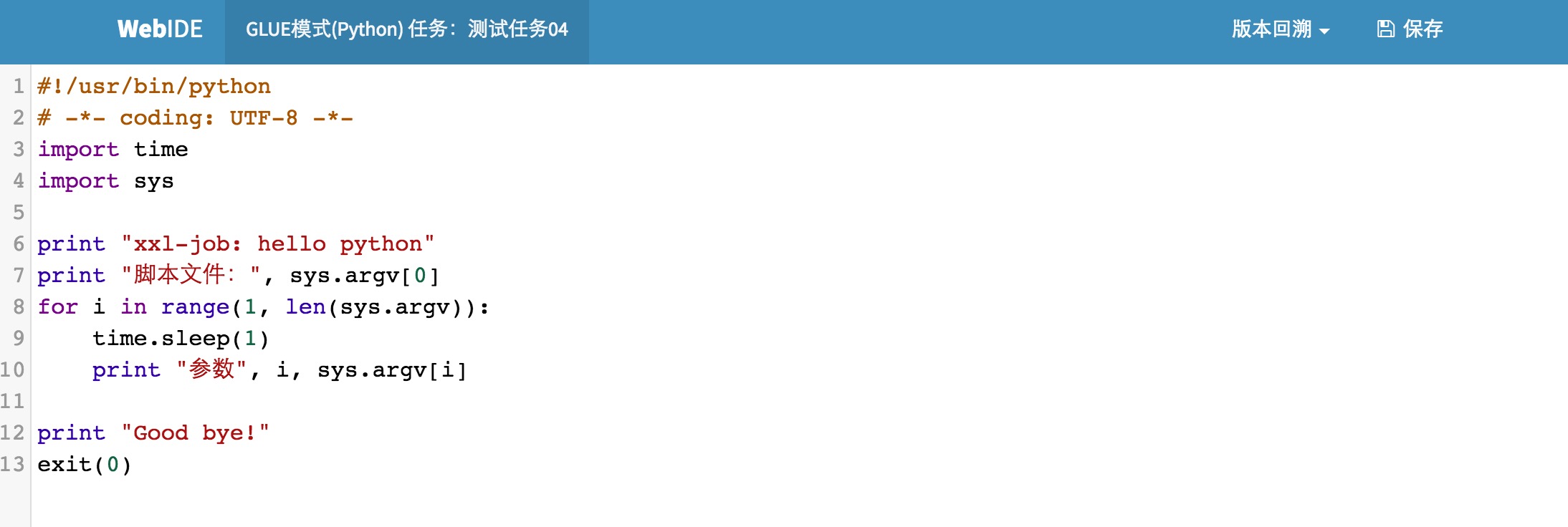
[**步骤一：调度中心，新建调度任务**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%ef%bc%8c%e6%96%b0%e5%bb%ba%e8%b0%83%e5%ba%a6%e4%bb%bb%e5%8a%a1-1)

参考上文“配置属性详细说明”对新建的任务进行参数配置，运行模式选中 "GLUE模式(Python)"；

[**步骤二：开发任务代码：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e5%bc%80%e5%8f%91%e4%bb%bb%e5%8a%a1%e4%bb%a3%e7%a0%81%ef%bc%9a-2)

选中指定任务，点击该任务右侧“GLUE”按钮，将会前往GLUE任务的Web IDE界面，在该界面支持对任务代码进行开发（也可以在IDE中开发完成后，复制粘贴到编辑中）。

该模式的任务实际上是一段 "python" 脚本；



[**3.5 GLUE模式(NodeJS)**](http://www.xuxueli.com/xxl-job/#/?id=_35-glue%e6%a8%a1%e5%bc%8fnodejs)

[**步骤一：调度中心，新建调度任务**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%b8%80%ef%bc%9a%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83%ef%bc%8c%e6%96%b0%e5%bb%ba%e8%b0%83%e5%ba%a6%e4%bb%bb%e5%8a%a1-2)

参考上文“配置属性详细说明”对新建的任务进行参数配置，运行模式选中 "GLUE模式(NodeJS)"；

[**步骤二：开发任务代码：**](http://www.xuxueli.com/xxl-job/#/?id=%e6%ad%a5%e9%aa%a4%e4%ba%8c%ef%bc%9a%e5%bc%80%e5%8f%91%e4%bb%bb%e5%8a%a1%e4%bb%a3%e7%a0%81%ef%bc%9a-3)

选中指定任务，点击该任务右侧“GLUE”按钮，将会前往GLUE任务的Web IDE界面，在该界面支持对任务代码进行开发（也可以在IDE中开发完成后，复制粘贴到编辑中）。

该模式的任务实际上是一段 "nodejS" 脚本；

[**四、任务管理**](http://www.xuxueli.com/xxl-job/#/?id=%e5%9b%9b%e3%80%81%e4%bb%bb%e5%8a%a1%e7%ae%a1%e7%90%86)

[**4.0 配置执行器**](http://www.xuxueli.com/xxl-job/#/?id=_40-%e9%85%8d%e7%bd%ae%e6%89%a7%e8%a1%8c%e5%99%a8)

点击进入"执行器管理"界面, 如下图: 

1、"调度中心OnLine:"右侧显示在线的"调度中心"列表, 任务执行结束后, 将会以failover的模式进行回调调度中心通知执行结果, 避免回调的单点风险;

2、"执行器列表" 中显示在线的执行器列表, 可通过"OnLine 机器"查看对应执行器的集群机器。

点击按钮 "+新增执行器" 弹框如下图, 可新增执行器配置: 

[**执行器属性说明**](http://www.xuxueli.com/xxl-job/#/?id=%e6%89%a7%e8%a1%8c%e5%99%a8%e5%b1%9e%e6%80%a7%e8%af%b4%e6%98%8e)

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；

[**4.1 新建任务**](http://www.xuxueli.com/xxl-job/#/?id=_41-%e6%96%b0%e5%bb%ba%e4%bb%bb%e5%8a%a1)

进入任务管理界面，点击“新增任务”按钮，在弹出的“新增任务”界面配置任务属性后保存即可。详情页参考章节 "三、任务详解"。

[**4.2 编辑任务**](http://www.xuxueli.com/xxl-job/#/?id=_42-%e7%bc%96%e8%be%91%e4%bb%bb%e5%8a%a1)

进入任务管理界面，选中指定任务。点击该任务右侧“编辑”按钮，在弹出的“编辑任务”界面更新任务属性后保存即可，可以修改设置的任务属性信息：

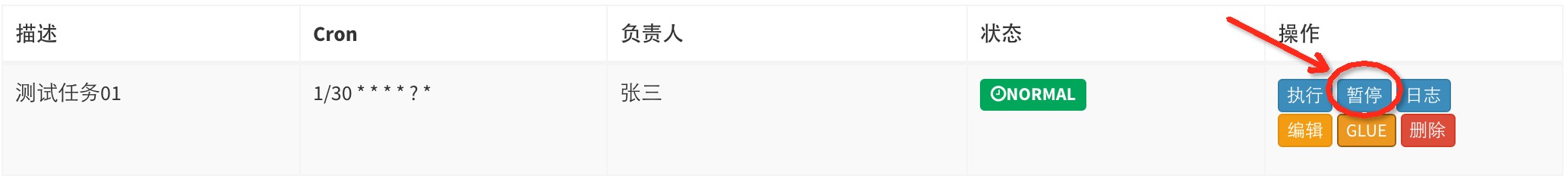
[**4.3 编辑GLUE代码**](http://www.xuxueli.com/xxl-job/#/?id=_43-%e7%bc%96%e8%be%91glue%e4%bb%a3%e7%a0%81)

该操作仅针对GLUE任务。

选中指定任务，点击该任务右侧“GLUE”按钮，将会前往GLUE任务的Web IDE界面，在该界面支持对任务代码进行开发。可参考章节 "3.2 GLUE模式(Java)"。

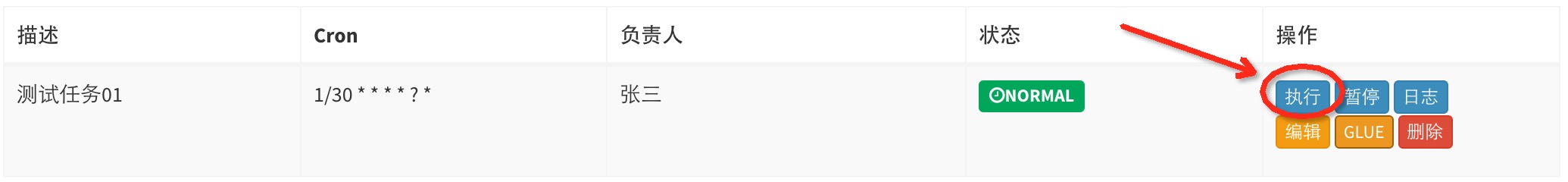
[**4.4 暂停/恢复任务**](http://www.xuxueli.com/xxl-job/#/?id=_44-%e6%9a%82%e5%81%9c%e6%81%a2%e5%a4%8d%e4%bb%bb%e5%8a%a1)

可对任务进行“暂停”和“恢复”操作。 需要注意的是，此处的暂停/恢复仅针对任务的后续调度触发行为，不会影响到已经触发的调度任务，如需终止已经触发的调度任务，可查看“4.8 终止运行中的任务”



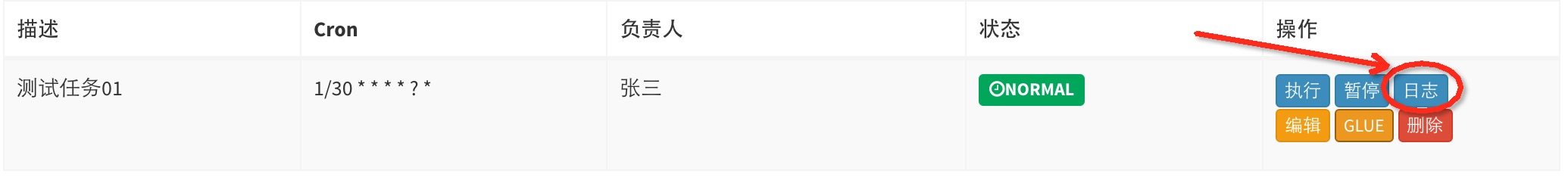
[**4.5 手动触发一次调度**](http://www.xuxueli.com/xxl-job/#/?id=_45-%e6%89%8b%e5%8a%a8%e8%a7%a6%e5%8f%91%e4%b8%80%e6%ac%a1%e8%b0%83%e5%ba%a6)

点击“执行”按钮，可手动触发一次任务调度，不影响原有调度规则。



[**4.6 查看调度日志**](http://www.xuxueli.com/xxl-job/#/?id=_46-%e6%9f%a5%e7%9c%8b%e8%b0%83%e5%ba%a6%e6%97%a5%e5%bf%97)

点击“日志”按钮，可以查看任务历史调度日志。在历史调入日志界面可查看每次任务调度的调度结果、执行结果等，点击“执行日志”按钮可查看执行器完整日志。





调度时间："调度中心"触发本次调度并向"执行器"发送任务执行信号的时间；

调度结果："调度中心"触发本次调度的结果，200表示成功，500或其他表示失败；

调度备注："调度中心"触发本次调度的日志信息；

执行器地址：本次任务执行的机器地址

运行模式：触发调度时任务的运行模式，运行模式可参考章节 "三、任务详解"；

任务参数：本地任务执行的入参

执行时间："执行器"中本次任务执行结束后回调的时间；

执行结果："执行器"中本次任务执行的结果，200表示成功，500或其他表示失败；

执行备注："执行器"中本次任务执行的日志信息；

操作：

"执行日志"按钮：点击可查看本地任务执行的详细日志信息；详见“4.7 查看执行日志”；

"终止任务"按钮：点击可终止本地调度对应执行器上本任务的执行线程，包括未执行的阻塞任务一并被终止；

[**4.7 查看执行日志**](http://www.xuxueli.com/xxl-job/#/?id=_47-%e6%9f%a5%e7%9c%8b%e6%89%a7%e8%a1%8c%e6%97%a5%e5%bf%97)

点击执行日志右侧的 “执行日志” 按钮，可跳转至执行日志界面，可以查看业务代码中打印的完整日志，如下图；



[**4.8 终止运行中的任务**](http://www.xuxueli.com/xxl-job/#/?id=_48-%e7%bb%88%e6%ad%a2%e8%bf%90%e8%a1%8c%e4%b8%ad%e7%9a%84%e4%bb%bb%e5%8a%a1)

仅针对执行中的任务。 在任务日志界面，点击右侧的“终止任务”按钮，将会向本次任务对应的执行器发送任务终止请求，将会终止掉本次任务，同时会清空掉整个任务执行队列。



任务终止时通过 "interrupt" 执行线程的方式实现, 将会触发 "InterruptedException" 异常。因此如果JobHandler内部catch到了该异常并消化掉的话, 任务终止功能将不可用。

因此, 如果遇到上述任务终止不可用的情况, 需要在JobHandler中应该针对 "InterruptedException" 异常进行特殊处理 (向上抛出) , 正确逻辑如下:

try{

// TODO

} catch (Exception e) {

if (e instanceof InterruptedException) {

throw e;

}

logger.warn("{}", e);

}

而且，在JobHandler中开启子线程时，子线程也不可catch处理"InterruptedException"，应该主动向上抛出。

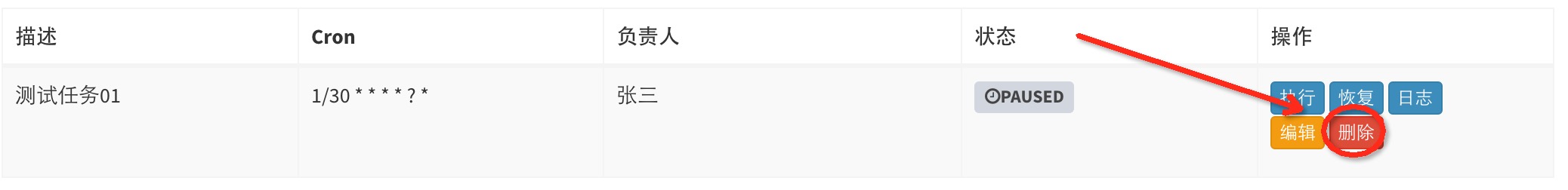
[**4.9 删除执行日志**](http://www.xuxueli.com/xxl-job/#/?id=_49-%e5%88%a0%e9%99%a4%e6%89%a7%e8%a1%8c%e6%97%a5%e5%bf%97)

在任务日志界面，选中执行器和任务之后，点击右侧的"删除"按钮将会出现"日志清理"弹框，弹框中支持选择不同类型的日志清理策略，选中后点击"确定"按钮即可进行日志清理操作； 



[**4.10 删除任务**](http://www.xuxueli.com/xxl-job/#/?id=_410-%e5%88%a0%e9%99%a4%e4%bb%bb%e5%8a%a1)

点击删除按钮，可以删除对应任务。



[**五、总体设计**](http://www.xuxueli.com/xxl-job/#/?id=%e4%ba%94%e3%80%81%e6%80%bb%e4%bd%93%e8%ae%be%e8%ae%a1)

[**5.1 源码目录介绍**](http://www.xuxueli.com/xxl-job/#/?id=_51-%e6%ba%90%e7%a0%81%e7%9b%ae%e5%bd%95%e4%bb%8b%e7%bb%8d)

- /doc :文档资料

- /db :“调度数据库”建表脚本

- /xxl-job-admin :调度中心，项目源码

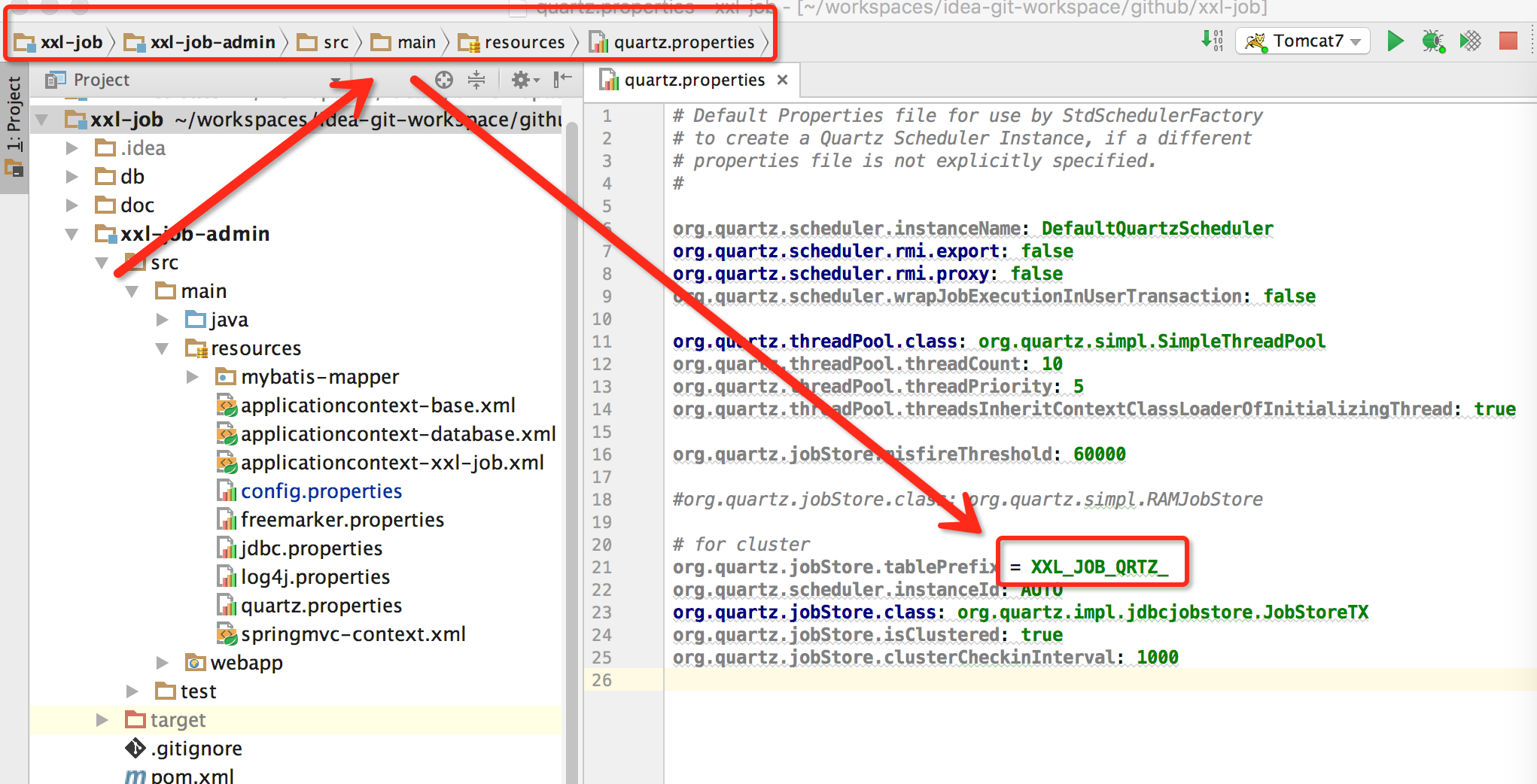
- /xxl-job-core :公共Jar依赖

- /xxl-job-executor-samples :执行器，Sample示例项目（大家可以在该项目上进行开发，也可以将现有项目改造生成执行器项目）

[**5.2 “调度数据库”配置**](http://www.xuxueli.com/xxl-job/#/?id=_52-%e8%b0%83%e5%ba%a6%e6%95%b0%e6%8d%ae%e5%ba%93%e9%85%8d%e7%bd%ae)

XXL-JOB调度模块基于Quartz集群实现，其“调度数据库”是在Quartz的11张集群mysql表基础上扩展而成。

XXL-JOB首先定制了Quartz原生表结构前缀（XXL\_JOB*QRTZ*）。



然后，在此基础上新增了几张张扩展表，如下：

- XXL\_JOB\_QRTZ\_TRIGGER\_GROUP：执行器信息表，维护任务执行器信息；

- XXL\_JOB\_QRTZ\_TRIGGER\_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\_QRTZ\_TRIGGER\_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\_QRTZ\_TRIGGER\_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\_QRTZ\_TRIGGER\_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；

因此，XXL-JOB调度数据库共计用于16张数据库表。

[**5.3 架构设计**](http://www.xuxueli.com/xxl-job/#/?id=_53-%e6%9e%b6%e6%9e%84%e8%ae%be%e8%ae%a1)

[**5.3.1 设计思想**](http://www.xuxueli.com/xxl-job/#/?id=_531-%e8%ae%be%e8%ae%a1%e6%80%9d%e6%83%b3)

将调度行为抽象形成“调度中心”公共平台，而平台自身并不承担业务逻辑，“调度中心”负责发起调度请求。

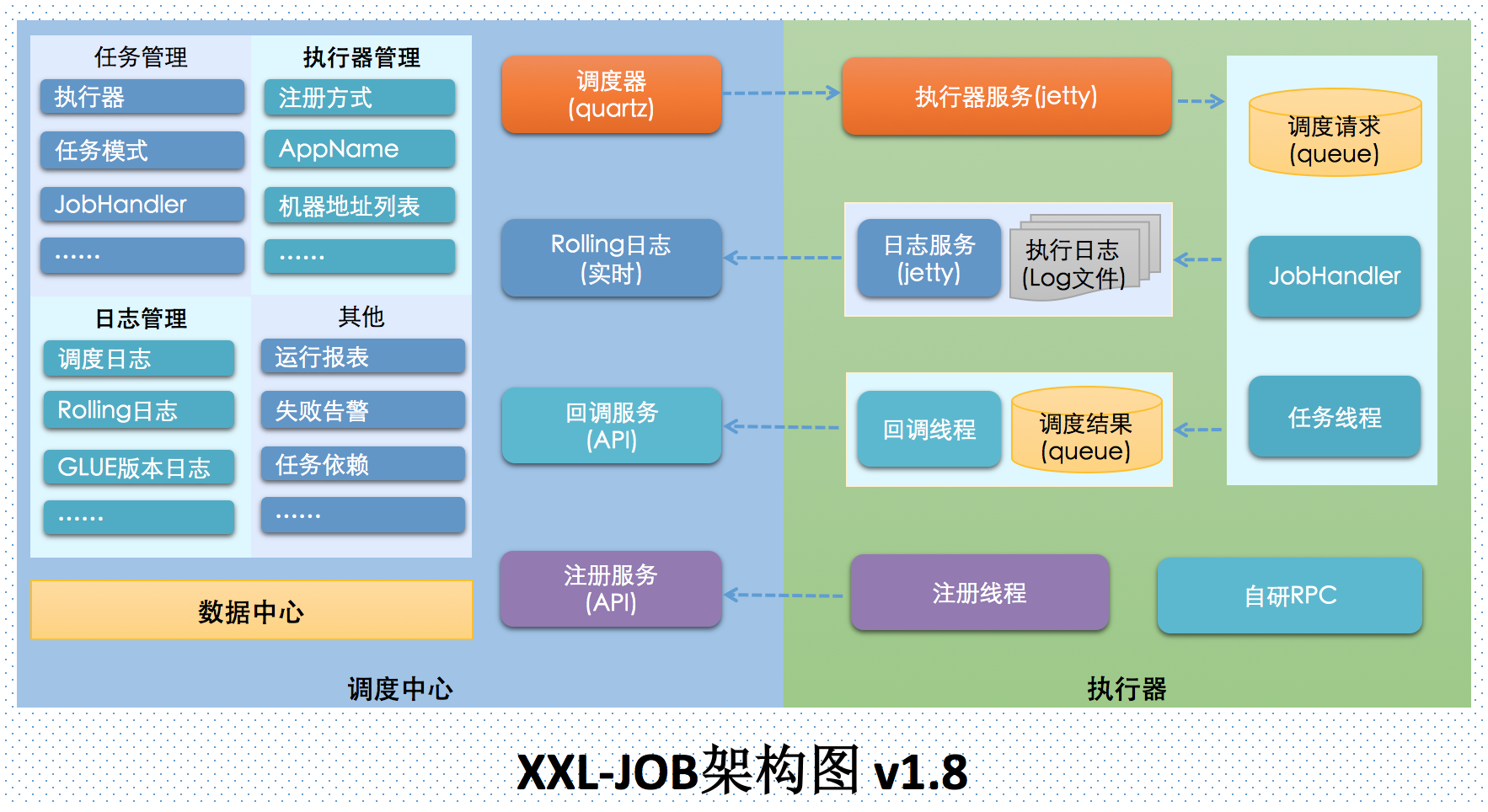
将任务抽象成分散的JobHandler，交由“执行器”统一管理，“执行器”负责接收调度请求并执行对应的JobHandler中业务逻辑。

因此，“调度”和“任务”两部分可以相互解耦，提高系统整体稳定性和扩展性；

[**5.3.2 系统组成**](http://www.xuxueli.com/xxl-job/#/?id=_532-%e7%b3%bb%e7%bb%9f%e7%bb%84%e6%88%90)

* **调度模块（调度中心）**： 负责管理调度信息，按照调度配置发出调度请求，自身不承担业务代码。调度系统与任务解耦，提高了系统可用性和稳定性，同时调度系统性能不再受限于任务模块； 支持可视化、简单且动态的管理调度信息，包括任务新建，更新，删除，GLUE开发和任务报警等，所有上述操作都会实时生效，同时支持监控调度结果以及执行日志，支持执行器Failover。
* **执行模块（执行器）**： 负责接收调度请求并执行任务逻辑。任务模块专注于任务的执行等操作，开发和维护更加简单和高效； 接收“调度中心”的执行请求、终止请求和日志请求等。

[**5.3.3 架构图**](http://www.xuxueli.com/xxl-job/#/?id=_533-%e6%9e%b6%e6%9e%84%e5%9b%be)



[**5.4 调度模块剖析**](http://www.xuxueli.com/xxl-job/#/?id=_54-%e8%b0%83%e5%ba%a6%e6%a8%a1%e5%9d%97%e5%89%96%e6%9e%90)

[**5.4.1 quartz的不足**](http://www.xuxueli.com/xxl-job/#/?id=_541-quartz%e7%9a%84%e4%b8%8d%e8%b6%b3)

Quartz作为开源作业调度中的佼佼者，是作业调度的首选。但是集群环境中Quartz采用API的方式对任务进行管理，从而可以避免上述问题，但是同样存在以下问题：

- 问题一：调用API的的方式操作任务，不人性化；

- 问题二：需要持久化业务QuartzJobBean到底层数据表中，系统侵入性相当严重。

- 问题三：调度逻辑和QuartzJobBean耦合在同一个项目中，这将导致一个问题，在调度任务数量逐渐增多，同时调度任务逻辑逐渐加重的情况加，此时调度系统的性能将大大受限于业务；

XXL-JOB弥补了quartz的上述不足之处。

[**5.4.2 RemoteHttpJobBean**](http://www.xuxueli.com/xxl-job/#/?id=_542-remotehttpjobbean)

常规Quartz的开发，任务逻辑一般维护在QuartzJobBean中，耦合很严重。XXL-JOB中“调度模块”和“任务模块”完全解耦，调度模块中的所有调度任务使用同一个QuartzJobBean，即RemoteHttpJobBean。不同的调度任务将各自参数维护在各自扩展表数据中，当触发RemoteHttpJobBean执行时，将会解析不同的任务参数发起远程调用，调用各自的远程执行器服务。

这种调用模型类似RPC调用，RemoteHttpJobBean提供调用代理的功能，而执行器提供远程服务的功能。

[**5.4.3 调度中心HA（集群）**](http://www.xuxueli.com/xxl-job/#/?id=_543-%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83ha%ef%bc%88%e9%9b%86%e7%be%a4%ef%bc%89)

基于Quartz的集群方案，数据库选用Mysql；集群分布式并发环境中使用QUARTZ定时任务调度，会在各个节点会上报任务，存到数据库中，执行时会从数据库中取出触发器来执行，如果触发器的名称和执行时间相同，则只有一个节点去执行此任务。

# for cluster

org.quartz.jobStore.tablePrefix = XXL\_JOB\_QRTZ\_

org.quartz.scheduler.instanceId: AUTO

org.quartz.jobStore.class: org.quartz.impl.jdbcjobstore.JobStoreTX

org.quartz.jobStore.isClustered: true

org.quartz.jobStore.clusterCheckinInterval: 1000

[**5.4.4 调度线程池**](http://www.xuxueli.com/xxl-job/#/?id=_544-%e8%b0%83%e5%ba%a6%e7%ba%bf%e7%a8%8b%e6%b1%a0)

默认线程池中线程的数量为10个，避免单线程因阻塞而引起任务调度延迟。

org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount: 10

org.quartz.threadPool.threadPriority: 5

org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true

XXL-JOB系统中业务逻辑在远程执行器执行，调度中心每次调度仅仅负责一次调度请求，执行器会将请求存入执行队列并且立即响应调度中心；相比直接在quartz的QuartzJobBean中执行业务逻辑，差别就像大象和羽毛；

XXL-JOB调度中心中每个JOB逻辑非常 “轻”，单个JOB一次运行平均耗时基本在 "100ms" 之内（基本是网络开销）；因此，可以保证使用有限的线程支撑大量的JOB并发运行；上面配置的10个线程至少可以支撑100个JOB正常运行；

[**5.4.5 @DisallowConcurrentExecution**](http://www.xuxueli.com/xxl-job/#/?id=_545-disallowconcurrentexecution)

XXL-JOB调度模块的“调度中心”默认不使用该注解，即默认开启并行机制，因为RemoteHttpJobBean为公共QuartzJobBean，这样在多线程调度的情况下，调度模块被阻塞的几率很低，大大提高了调度系统的承载量。

XXL-JOB的每个调度任务虽然在调度模块是并行调度执行的，但是任务调度传递到任务模块的“执行器”确实串行执行的，同时支持任务终止。

[**5.4.6 misfire**](http://www.xuxueli.com/xxl-job/#/?id=_546-misfire)

错过了触发时间，处理规则。 可能原因：服务重启；调度线程被QuartzJobBean阻塞，线程被耗尽；某个任务启用了@DisallowConcurrentExecution，上次调度持续阻塞，下次调度被错过；

quartz.properties中关于misfire的阀值配置如下，单位毫秒：

org.quartz.jobStore.misfireThreshold: 60000

Misfire规则： withMisfireHandlingInstructionDoNothing：不触发立即执行，等待下次调度； withMisfireHandlingInstructionIgnoreMisfires：以错过的第一个频率时间立刻开始执行； withMisfireHandlingInstructionFireAndProceed：以当前时间为触发频率立刻触发一次执行；

XXL-JOB默认misfire规则为：withMisfireHandlingInstructionDoNothing

CronScheduleBuilder cronScheduleBuilder = CronScheduleBuilder.cronSchedule(jobInfo.getJobCron()).withMisfireHandlingInstructionDoNothing();

CronTrigger cronTrigger = TriggerBuilder.newTrigger().withIdentity(triggerKey).withSchedule(cronScheduleBuilder).build();

[**5.4.7 日志回调服务**](http://www.xuxueli.com/xxl-job/#/?id=_547-%e6%97%a5%e5%bf%97%e5%9b%9e%e8%b0%83%e6%9c%8d%e5%8a%a1)

调度模块的“调度中心”作为Web服务部署时，一方面承担调度中心功能，另一方面也为执行器提供API服务。

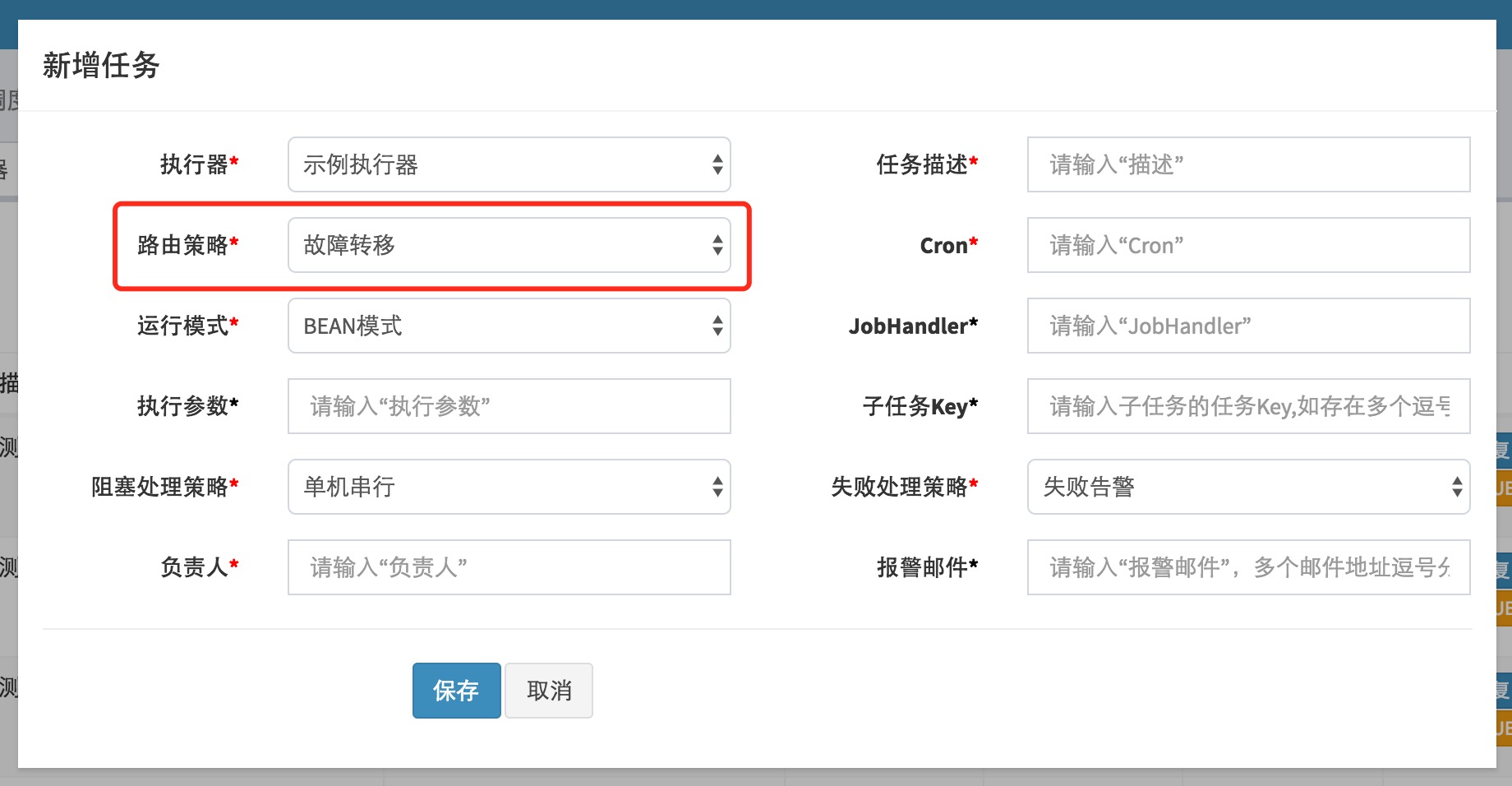
调度中心提供的"日志回调服务API服务"代码位置如下：

xxl-job-admin#com.xxl.job.admin.controller.JobApiController.callback

“执行器”在接收到任务执行请求后，执行任务，在执行结束之后会将执行结果回调通知“调度中心”：

[**5.4.8 任务HA（Failover）**](http://www.xuxueli.com/xxl-job/#/?id=_548-%e4%bb%bb%e5%8a%a1ha%ef%bc%88failover%ef%bc%89)

执行器如若集群部署，调度中心将会感知到在线的所有执行器，如“127.0.0.1:9997, 127.0.0.1:9998, 127.0.0.1:9999”。

当任务"路由策略"选择"故障转移(FAILOVER)"时，当调度中心每次发起调度请求时，会按照顺序对执行器发出心跳检测请求，第一个检测为存活状态的执行器将会被选定并发送调度请求。 

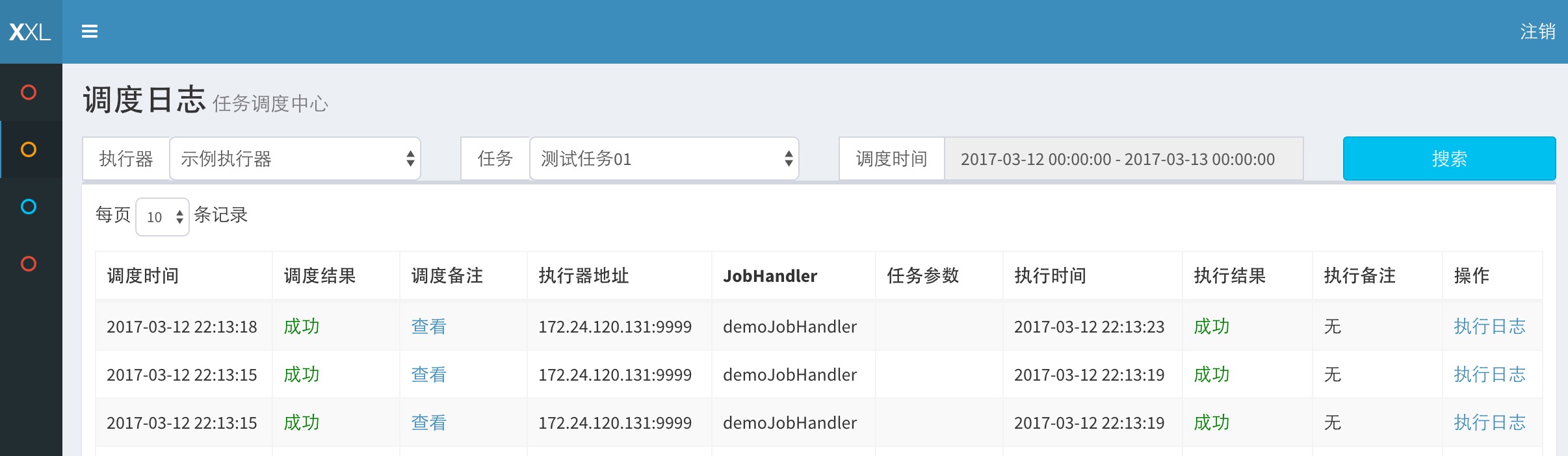
调度成功后，可在日志监控界面查看“调度备注”，如下； 

“调度备注”可以看出本地调度运行轨迹，执行器的"注册方式"、"地址列表"和任务的"路由策略"。"故障转移(FAILOVER)"路由策略下，调度中心首先对第一个地址进行心跳检测，心跳失败因此自动跳过，第二个依然心跳检测失败…… 直至心跳检测第三个地址“127.0.0.1:9999”成功，选定为“目标执行器”；然后对“目标执行器”发送调度请求，调度流程结束，等待执行器回调执行结果。

[**5.4.9 调度日志**](http://www.xuxueli.com/xxl-job/#/?id=_549-%e8%b0%83%e5%ba%a6%e6%97%a5%e5%bf%97)

调度中心每次进行任务调度，都会记录一条任务日志，任务日志主要包括以下三部分内容：

* 任务信息：包括“执行器地址”、“JobHandler”和“执行参数”等属性，根据这些参数，可以精确的定位任务执行的具体机器和任务代码；
* 调度信息：包括“调度时间”、“调度结果”和“调度日志”等，根据这些参数，可以了解“调度中心”发起调度请求时具体情况。
* 执行信息：包括“执行时间”、“执行结果”和“执行日志”等，根据这些参数，可以了解在“执行器”端任务执行的具体情况；



调度日志，针对单次调度，属性说明如下：

* 执行器地址：任务执行的机器地址；
* JobHandler：Bean模式表示任务执行的JobHandler名称；
* 任务参数：任务执行的入参；
* 调度时间：调度中心，发起调度的时间；
* 调度结果：调度中心，发起调度的结果，SUCCESS或FAIL；
* 调度备注：调度中心，发起调度的备注信息，如地址心跳检测日志等；
* 执行时间：执行器，任务执行结束后回调的时间；
* 执行结果：执行器，任务执行的结果，SUCCESS或FAIL；
* 执行备注：执行器，任务执行的备注信息，如异常日志等；
* 执行日志：任务执行过程中，业务代码中打印的完整执行日志，见“4.7 查看执行日志”；

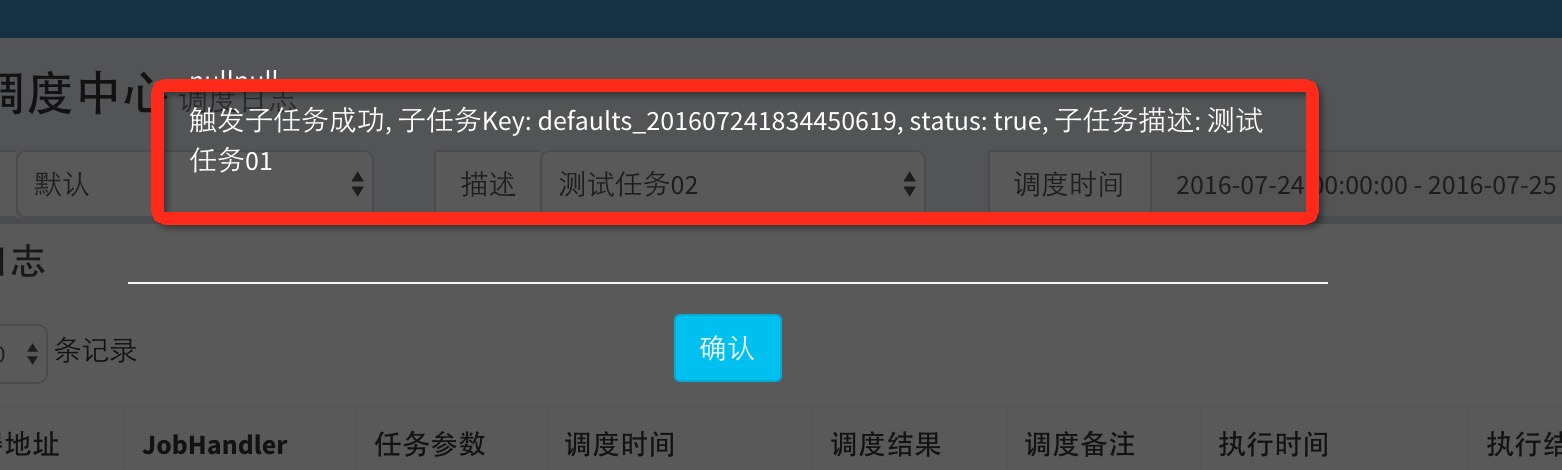
[**5.4.10 任务依赖**](http://www.xuxueli.com/xxl-job/#/?id=_5410-%e4%bb%bb%e5%8a%a1%e4%be%9d%e8%b5%96)

原理：XXL-JOB中每个任务都对应有一个任务Key，同时，每个任务支持设置属性“子任务Key”，因此，通过“任务Key”可以匹配任务依赖关系。

当父任务执行结束并且执行成功时，将会根据“子任务Key”匹配子任务依赖，如果匹配到子任务，将会主动触发一次子任务的执行。

在任务日志界面，点击任务的“执行备注”的“查看”按钮，可以看到匹配子任务以及触发子任务执行的日志信息，如无信息则表示未触发子任务执行，可参考下图。





[**5.5 任务 "运行模式" 剖析**](http://www.xuxueli.com/xxl-job/#/?id=_55-%e4%bb%bb%e5%8a%a1-quot%e8%bf%90%e8%a1%8c%e6%a8%a1%e5%bc%8fquot-%e5%89%96%e6%9e%90)

[**5.5.1 "Bean模式" 任务**](http://www.xuxueli.com/xxl-job/#/?id=_551-quotbean%e6%a8%a1%e5%bc%8fquot-%e4%bb%bb%e5%8a%a1)

开发步骤：可参考 "章节三" ； 原理：每个Bean模式任务都是一个Spring的Bean类实例，它被维护在“执行器”项目的Spring容器中。任务类需要加“@JobHander(value="名称")”注解，因为“执行器”会根据该注解识别Spring容器中的任务。任务类需要继承统一接口“IJobHandler”，任务逻辑在execute方法中开发，因为“执行器”在接收到调度中心的调度请求时，将会调用“IJobHandler”的execute方法，执行任务逻辑。

[**5.5.2 "GLUE模式(Java)" 任务**](http://www.xuxueli.com/xxl-job/#/?id=_552-quotglue%e6%a8%a1%e5%bc%8fjavaquot-%e4%bb%bb%e5%8a%a1)

开发步骤：可参考 "章节三" ； 原理：每个 "GLUE模式(Java)" 任务的代码，实际上是“一个继承自“IJobHandler”的实现类的类代码”，“执行器”接收到“调度中心”的调度请求时，会通过Groovy类加载器加载此代码，实例化成Java对象，同时注入此代码中声明的Spring服务（请确保Glue代码中的服务和类引用在“执行器”项目中存在），然后调用该对象的execute方法，执行任务逻辑。

[**5.5.3 GLUE模式(Shell) + GLUE模式(Python) + GLUE模式(NodeJS)**](http://www.xuxueli.com/xxl-job/#/?id=_553-glue%e6%a8%a1%e5%bc%8fshell-glue%e6%a8%a1%e5%bc%8fpython-glue%e6%a8%a1%e5%bc%8fnodejs)

开发步骤：可参考 "章节三" ； 原理：脚本任务的源码托管在调度中心，脚本逻辑在执行器运行。当触发脚本任务时，执行器会加载脚本源码在执行器机器上生成一份脚本文件，然后通过Java代码调用该脚本；并且实时将脚本输出日志写到任务日志文件中，从而在调度中心可以实时监控脚本运行情况；脚本返回码为0时表示执行成功，其他标示执行失败。

目前支持的脚本类型如下：

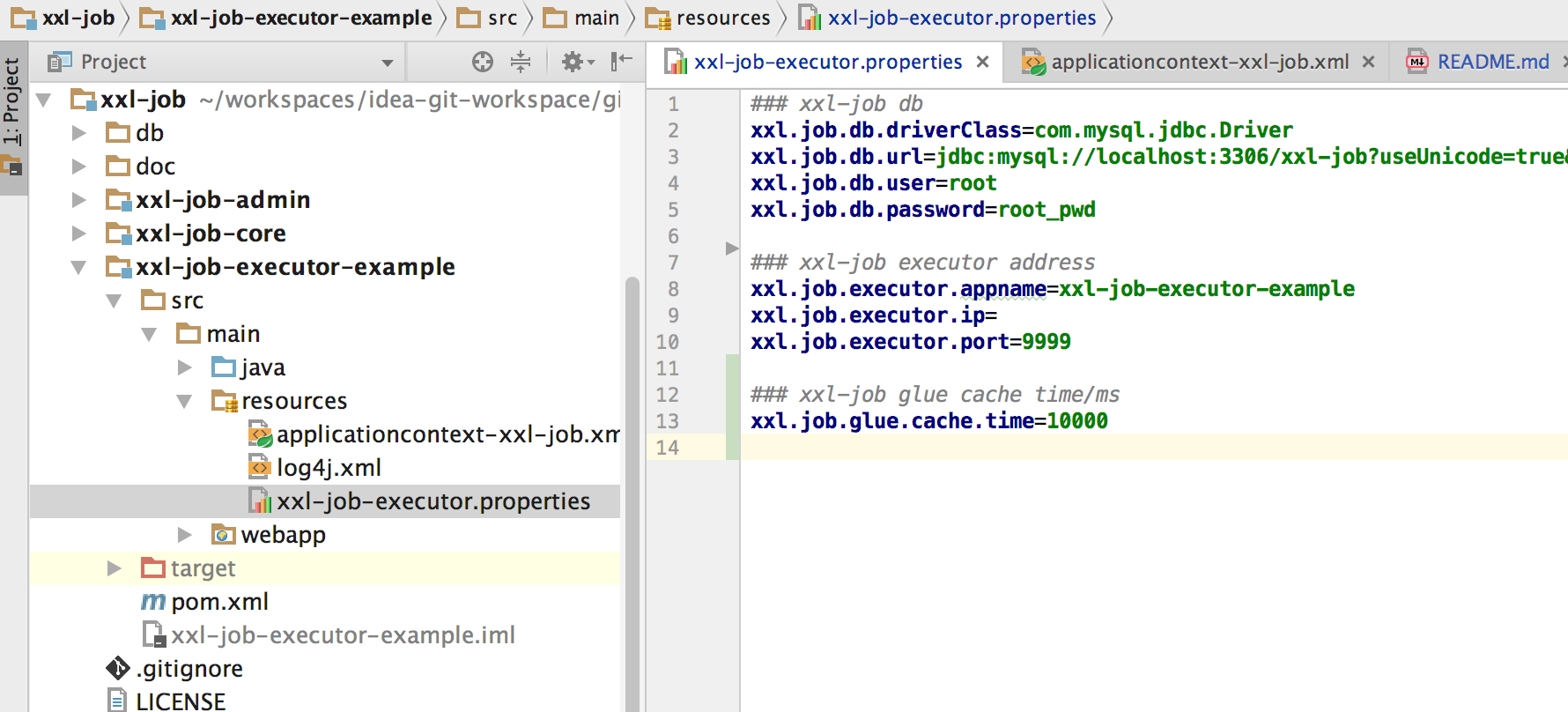
- shell脚本：任务运行模式选择为 "GLUE模式(Shell)"时支持 "shell" 脚本任务；

- python脚本：任务运行模式选择为 "GLUE模式(Python)"时支持 "python" 脚本任务；

- nodejs脚本：务运行模式选择为 "GLUE模式(NodeJS)"时支持 "nodejs" 脚本任务；

[**5.5.4 执行器**](http://www.xuxueli.com/xxl-job/#/?id=_554-%e6%89%a7%e8%a1%8c%e5%99%a8)

执行器实际上是一个内嵌的Jetty服务器，默认端口9999，如下图配置文件所示（参数：xxl.job.executor.port）。



在项目启动时，执行器会通过“@JobHander”识别Spring容器中“Bean模式任务”，以注解的value属性为key管理起来。

“执行器”接收到“调度中心”的调度请求时，如果任务类型为“Bean模式”，将会匹配Spring容器中的“Bean模式任务”，然后调用其execute方法，执行任务逻辑。如果任务类型为“GLUE模式”，将会加载GLue代码，实例化Java对象，注入依赖的Spring服务（注意：Glue代码中注入的Spring服务，必须存在与该“执行器”项目的Spring容器中），然后调用execute方法，执行任务逻辑。

[**5.5.5 任务日志**](http://www.xuxueli.com/xxl-job/#/?id=_555-%e4%bb%bb%e5%8a%a1%e6%97%a5%e5%bf%97)

XXL-JOB会为每次调度请求生成一个单独的日志文件，需要通过 "XxlJobLogger.log" 打印执行日志，“调度中心”查看执行日志时将会加载对应的日志文件。

(历史版本通过重写LOG4J的Appender实现，存在依赖限制，该方式在新版本已经被抛弃)

日志文件存放的位置可在“执行器”配置文件进行自定义，默认目录格式为：/data/applogs/xxl-job/jobhandler/“格式化日期”/“数据库调度日志记录的主键ID.log”。

在JobHandler中开启子线程时，子线程将会将会把日志打印在父线程即JobHandler的执行日志中，方便日志追踪。

[**5.6 通讯模块剖析**](http://www.xuxueli.com/xxl-job/#/?id=_56-%e9%80%9a%e8%ae%af%e6%a8%a1%e5%9d%97%e5%89%96%e6%9e%90)

[**5.6.1 一次完整的任务调度通讯流程**](http://www.xuxueli.com/xxl-job/#/?id=_561-%e4%b8%80%e6%ac%a1%e5%ae%8c%e6%95%b4%e7%9a%84%e4%bb%bb%e5%8a%a1%e8%b0%83%e5%ba%a6%e9%80%9a%e8%ae%af%e6%b5%81%e7%a8%8b)

- 1、“调度中心”向“执行器”发送http调度请求: “执行器”中接收请求的服务，实际上是一台内嵌jetty服务器，默认端口9999;

- 2、“执行器”执行任务逻辑；

- 3、“执行器”http回调“调度中心”调度结果: “调度中心”中接收回调的服务，是针对执行器开放一套API服务;

[**5.6.2 通讯数据加密**](http://www.xuxueli.com/xxl-job/#/?id=_562-%e9%80%9a%e8%ae%af%e6%95%b0%e6%8d%ae%e5%8a%a0%e5%af%86)

调度中心向执行器发送的调度请求时使用RequestModel和ResponseModel两个对象封装调度请求参数和响应数据, 在进行通讯之前底层会将上述两个对象对象序列化，并进行数据协议以及时间戳检验,从而达到数据加密的功能;

[**5.7 任务注册, 任务自动发现**](http://www.xuxueli.com/xxl-job/#/?id=_57-%e4%bb%bb%e5%8a%a1%e6%b3%a8%e5%86%8c-%e4%bb%bb%e5%8a%a1%e8%87%aa%e5%8a%a8%e5%8f%91%e7%8e%b0)

自v1.5版本之后, 任务取消了"任务执行机器"属性, 改为通过任务注册和自动发现的方式, 动态获取远程执行器地址并执行。

AppName: 每个执行器机器集群的唯一标示, 任务注册以 "执行器" 为最小粒度进行注册; 每个任务通过其绑定的执行器可感知对应的执行器机器列表;

注册表: 见"XXL\_JOB\_QRTZ\_TRIGGER\_REGISTRY"表, "执行器" 在进行任务注册时将会周期性维护一条注册记录，即机器地址和AppName的绑定关系; "调度中心" 从而可以动态感知每个AppName在线的机器列表;

执行器注册: 任务注册Beat周期默认30s; 执行器以一倍Beat进行执行器注册, 调度中心以一倍Beat进行动态任务发现; 注册信息的失效时间被三倍Beat;

执行器注册摘除：执行器销毁时，将会主动上报调度中心并摘除对应的执行器机器信息，提高心跳注册的实时性；

为保证系统"轻量级"并且降低学习部署成本，没有采用Zookeeper作为注册中心，采用DB方式进行任务注册发现；

[**5.8 任务执行结果**](http://www.xuxueli.com/xxl-job/#/?id=_58-%e4%bb%bb%e5%8a%a1%e6%89%a7%e8%a1%8c%e7%bb%93%e6%9e%9c)

自v1.6.2之后，任务执行结果通过 "IJobHandler" 的返回值 "ReturnT" 进行判断； 当返回值符合 "ReturnT.code == ReturnT.SUCCESS\_CODE" 时表示任务执行成功，否则表示任务执行失败，而且可以通过 "ReturnT.msg" 回调错误信息给调度中心； 从而，在任务逻辑中可以方便的控制任务执行结果；

[**5.9 分片广播 & 动态分片**](http://www.xuxueli.com/xxl-job/#/?id=_59-%e5%88%86%e7%89%87%e5%b9%bf%e6%92%ad-amp-%e5%8a%a8%e6%80%81%e5%88%86%e7%89%87)

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对应集群中所有执行器执行一次任务，同时传递分片参数；可根据分片参数开发分片任务；

"分片广播" 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。

"分片广播" 和普通任务开发流程一致，不同之处在于可以可以获取分片参数，获取分片参数对象的代码如下（可参考Sample示例执行器中的示例任务"ShardingJobHandler" ）：

ShardingUtil.ShardingVO shardingVO = ShardingUtil.getShardingVo();

该分片参数对象拥有两个属性：

index：当前分片序号(从0开始)，执行器集群列表中当前执行器的序号；

total：总分片数，执行器集群的总机器数量；

该特性适用场景如：

* 1、分片任务场景：10个执行器的集群来处理10w条数据，每台机器只需要处理1w条数据，耗时降低10倍；
* 2、广播任务场景：广播执行器机器运行shell脚本、广播集群节点进行缓存更新等

[**5.10 访问令牌（AccessToken）**](http://www.xuxueli.com/xxl-job/#/?id=_510-%e8%ae%bf%e9%97%ae%e4%bb%a4%e7%89%8c%ef%bc%88accesstoken%ef%bc%89)

为提升系统安全性，调度中心和执行器进行安全性校验，双方AccessToken匹配才允许通讯；

调度中心和执行器，可通过配置项 "xxl.job.accessToken" 进行AccessToken的设置。

调度中心和执行器，如果需要正常通讯，只有两种设置；

* 设置一：调度中心和执行器，均不设置AccessToken；关闭安全性校验；
* 设置二：调度中心和执行器，设置了相同的AccessToken；

[**5.11 调度中心API服务**](http://www.xuxueli.com/xxl-job/#/?id=_511-%e8%b0%83%e5%ba%a6%e4%b8%ad%e5%bf%83api%e6%9c%8d%e5%8a%a1)

调度中心提供了API服务，供执行器和业务方选择使用，目前提供的API服务有：

1、任务结果回调服务；

2、执行器注册服务；

3、执行器注册摘除服务；

4、触发任务单次执行服务，支持任务根据业务事件触发；

调度中心API服务位置：com.xxl.job.core.biz.AdminBiz.java

调度中心API服务请求参考代码：com.xxl.job.dao.impl.AdminBizTest.java