XXL-JOB 1.3 用户手册

[一 简介 2](#_Toc28634)

[1.1 背景 2](#_Toc16823)

[1.2 特点 2](#_Toc1717)

[1.3 发展 2](#_Toc28585)

[1.4 下载 3](#_Toc5620)

[1.5 环境 3](#_Toc9420)

[二 快速入门 4](#_Toc17287)

[2.1 初始化“调度数据库” 4](#_Toc21562)

[2.2 编译源码 5](#_Toc12133)

[2.3 配置部署“调度中心” 5](#_Toc14924)

[2.4 配置部署“执行器项目” 7](#_Toc14292)

[2.5 开发第一个任务“Hello World” 8](#_Toc18995)

[三 任务详解 12](#_Toc289)

[4.1 BEAN模式任务 12](#_Toc9574)

[4.3 GLUE模式任务 13](#_Toc25562)

[四 任务管理 15](#_Toc21349)

[5.1 编辑任务信息 15](#_Toc15875)

[5.2 编辑GLUE代码 15](#_Toc25755)

[5.3 恢复/暂停 16](#_Toc18152)

[5.4 手动触发一次调度 16](#_Toc16755)

[5.5 查看日志 16](#_Toc17007)

[5.6 删除任务 17](#_Toc11984)

[5.7 终止运行中的任务 17](#_Toc6175)

[五 总体设计 19](#_Toc7049)

[5.1 源码目录介绍 19](#_Toc13110)

[5.2 “调度数据库”配置 19](#_Toc9890)

[5.3 架构设计 21](#_Toc31866)

[6.4 调度模块 21](#_Toc3147)

[6.5 执行模块 24](#_Toc953)

[6.6 通讯模块 26](#_Toc22521)

[七 其他 27](#_Toc24076)

[7.1 接入登记 27](#_Toc27660)

[7.2 报告问题 27](#_Toc10066)

# 一 简介

## 1.1 背景

Quartz作为开源作业调度中的佼佼者，是作业调度的首选。

非集群Quartz开发需要配置Trigger，JobBean，配置Corn等等，虽然简单，但是流程较多；集群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；

为了解决上述问题，期望是能够创造一种全新的调度体验。新系统目标是：更易维护、更高的可用性和更好的伸缩性。这要求“调度”和“作业”进行解耦，同时支持友好的可视化管理。由此，我萌发了打造XXL-JOB的想法，

## 1.2 特点

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、HA：“调度中心”基于集群Quartz实现，可保证调度中心HA；

4、一致性：“调度中心”通过DB锁保证集群分布式调度的一致性；

5、自定义任务参数：支持在线配置调度任务入参，即时生效；

6、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；

7、执行日志：支持在线查看调度结果，并且查看完整的执行日志；

8、邮件报警：任务失败时支持邮件报警，同时可自定义失败次数阀值；

9、支持登录验证；

## 1.3 发展

我于2015-11-28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2015-12-05日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@红薯的推荐，同期分别达到了OSCHINA的“热门动弹”排行第一和git.oschina的月热度排行第一，在此特别感谢红薯，感谢大家的关注和支持。

于2015-12月中旬我将XXL-JOB发表到我司内部知识库，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。

我司大众点评已接入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V1.3）。自2016-01-21接入至2016-05-20为止，该系统已调度40000余次，表现优异。

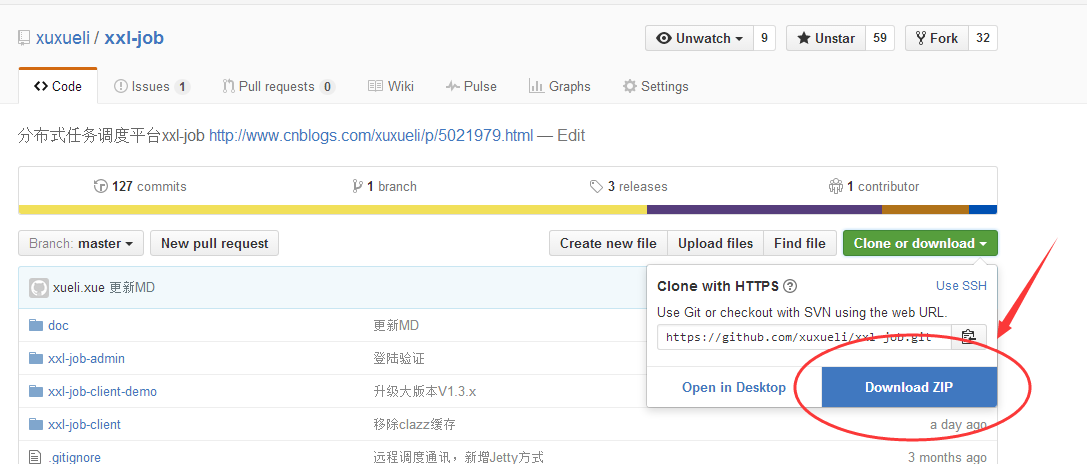
至今，XXL-JOB已接入多家公司的线上产品线，场景如电商业务，O2O业务和大数据作业等等，欢迎大家使用，XXL-JOB也将拥抱变化，持续发展。

## 1.4 下载

Github地址：<https://github.com/xuxueli/xxl-job>

Git@OSC地址：<http://git.oschina.net/xuxueli0323/xxl-job>

（我将会在两个git仓库同步发布最新代码）



（图1.3：github下载链接位置）

源码下载请前往github自行下载，下载位置见上图1.3；

## 1.5 环境

Maven3

JDK1.7

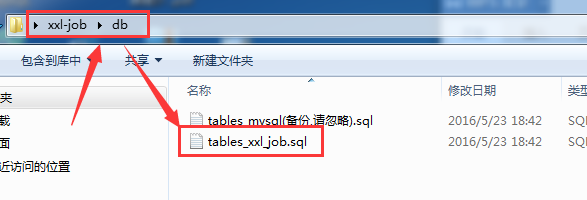
Tomcat7

Mysql5.5

# 二 快速入门

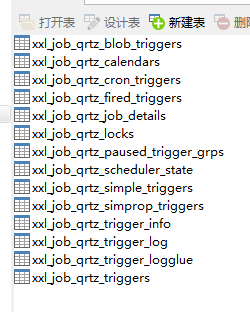
## 2.1 初始化“调度数据库”

请下载项目源码并解压，然后前往“图2.1A”所示目录，获取SQL脚本并执行，脚本文件位置：**“源码解压根目录”\xxl-job\db\tables\_xxl\_job.sql**



（图2.1A：数据库建表SQL文件位置截图，）

正常情况下，应该生成（图2.1B）中所示14张表。

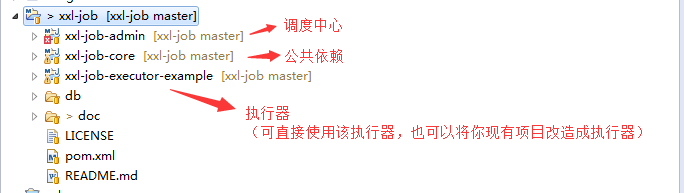


（图2.1B：调度数据库表一览）

## 2.2 编译源码

解压源码，按照maven格式将源码导入IDE（文档以Eclipse为例），更新项目pom依赖，maven编译项目。

正常情况下，项目结构应该如图2.2所示，



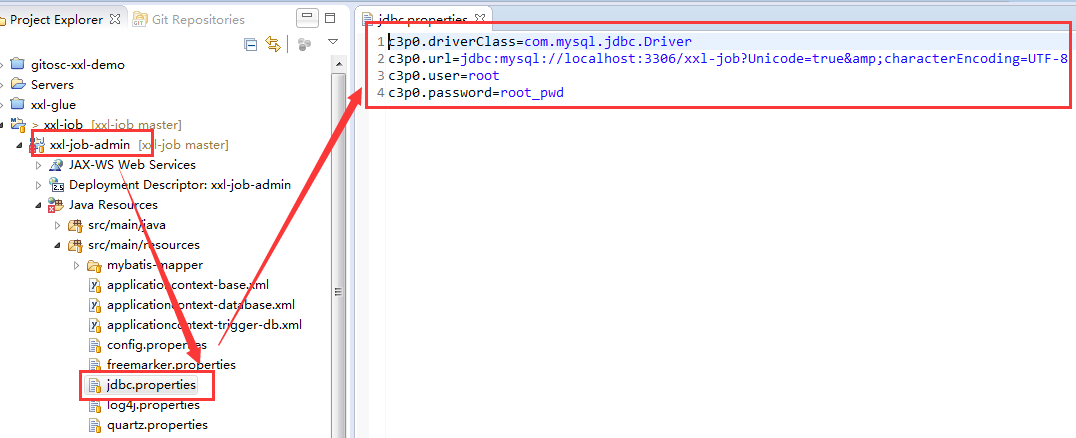
（图2.2：项目源码导入Eclipse截图）

## 2.3 配置部署“调度中心”

“调度中心”项目：xxl-job-admin

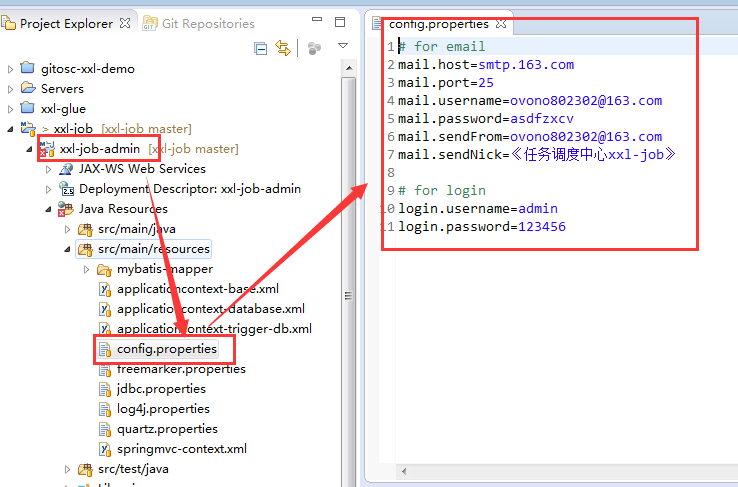
作用：统一管理任务调度平台上调度任务，负责触发调度执行。

**A：配置调度中心JDBC链接**：请在图2.2A所示位置配置jdbc链接地址，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。



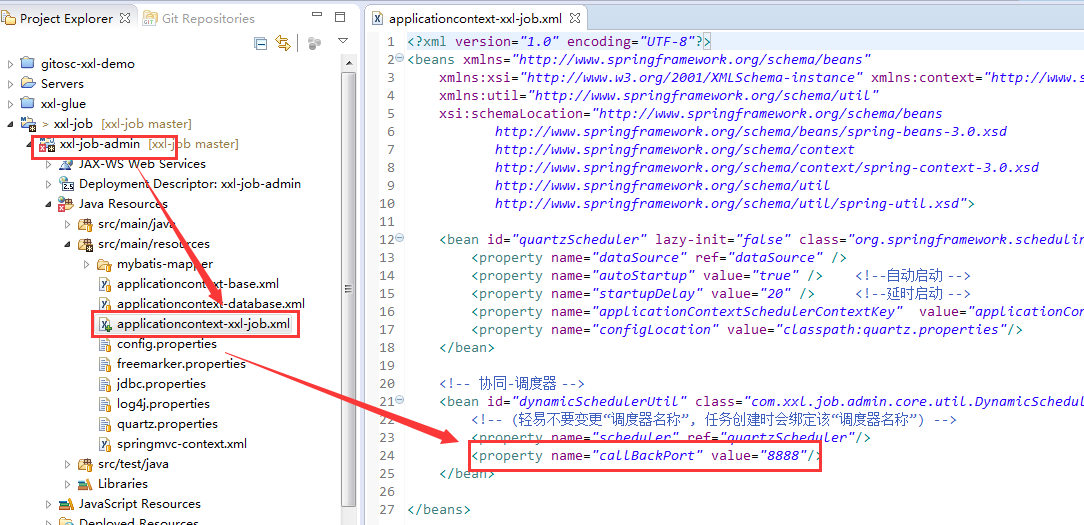
（图2.2A：数据库JDBC配置截图）

**B：配置报警邮箱和登陆账号**：请在图2.2B所示位置，设置自己的报警邮件发送邮箱和登陆账号密码。



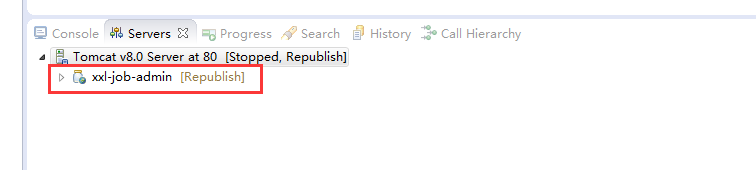
（图2.2B：系统配置截图）

**C：配置“调度中心”日志回调端口**：由于“调度中心”和“执行器”部署在不同机器，“执行器”会请求该端口回调通知任务执行情况。如图2.2C所示，默认回调服务端口号为8888。（此端口除非与现有端口冲突，可自行修改，否则请忽视）



（图2.2C）

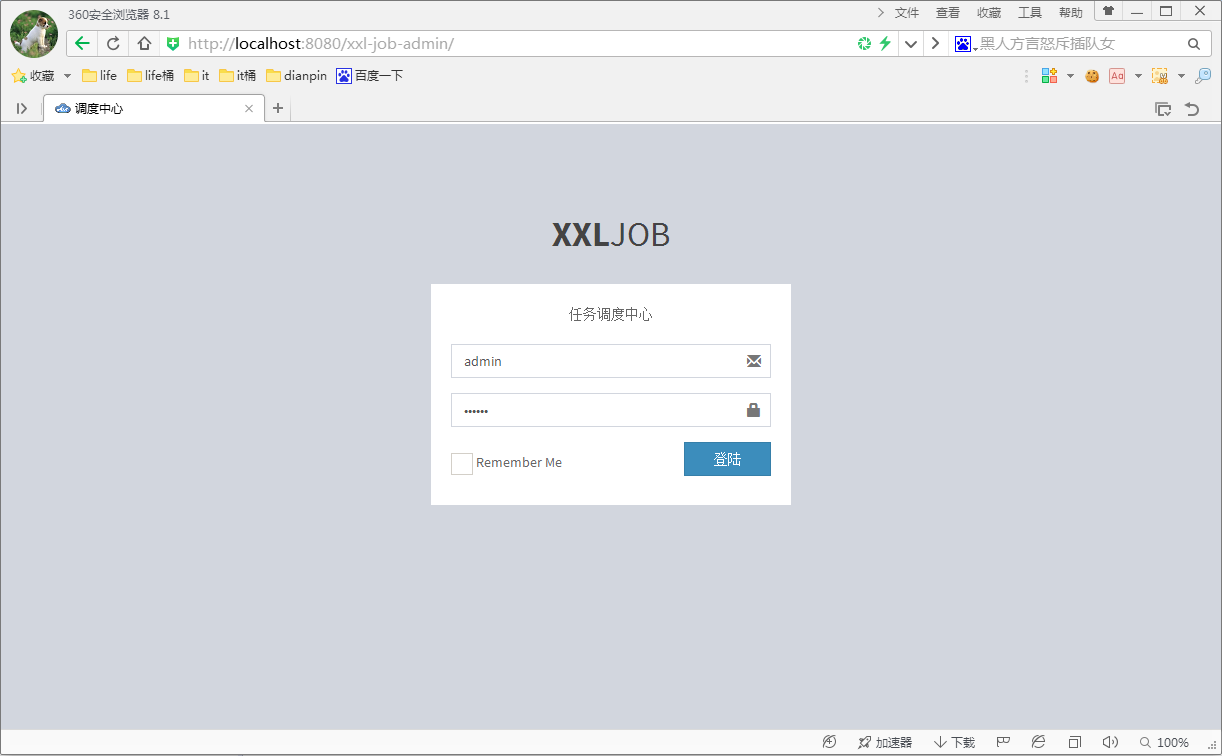
**部署项目**：如果已经正确进行上述配置，可将项目部署到eclipse下的tomcat服务器中，如图2.2D所示。或者，将“调度中心”项目导出war包单独部署。



（图2.2D：调度中心部署截图）

**访问链接**：http://localhost:8080/xxl-job-admin/ ，登陆界面如图2.2E所示。

至此“调度中心”项目已经部署结束。



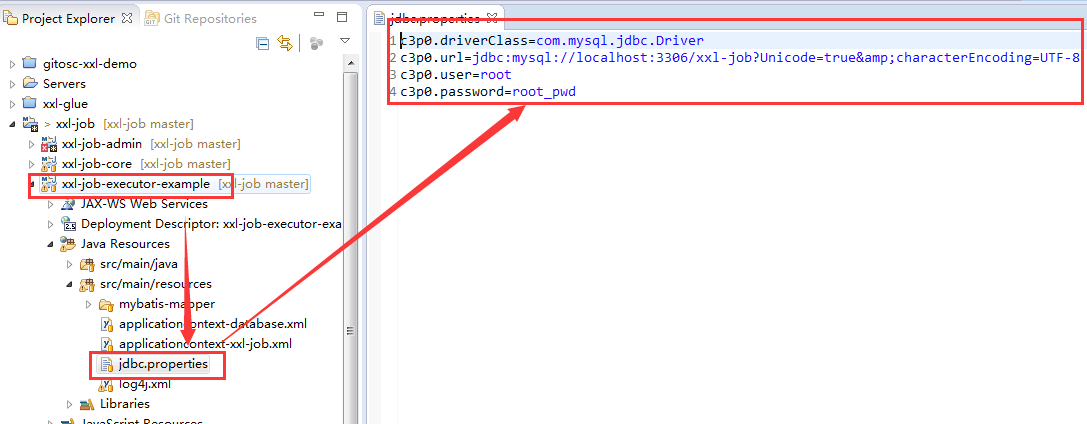
（图2.2E：“调度中心”登陆界面）

## 2.4 配置部署“执行器项目”

“执行器”项目：xxl-job-executor-example

作用：负责接收“调度中心”的调度并执行；

**A：配置Jdbc链接**：请在图2.4A所示位置配置jdbc链接地址，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。



（图2.4A：“执行器”项目Jdbc链接配置截图）

**B：配置“执行器”端口**：由于“调度中心”和“执行器”部署在不同机器上，“调度中心”会请求该端口触发任务执行。如图2.4B所示，默认的“执行器”端口是9999，如果与系统现有端口冲突可自行修改，如若不冲突，可忽略。



（图2.4B：“执行器”端口截图）

**部署项目**：如果已经正确进行上述配置，可将项目部署到eclipse下的tomcat服务器中。或者，将“执行器”项目导出war包单独部署。

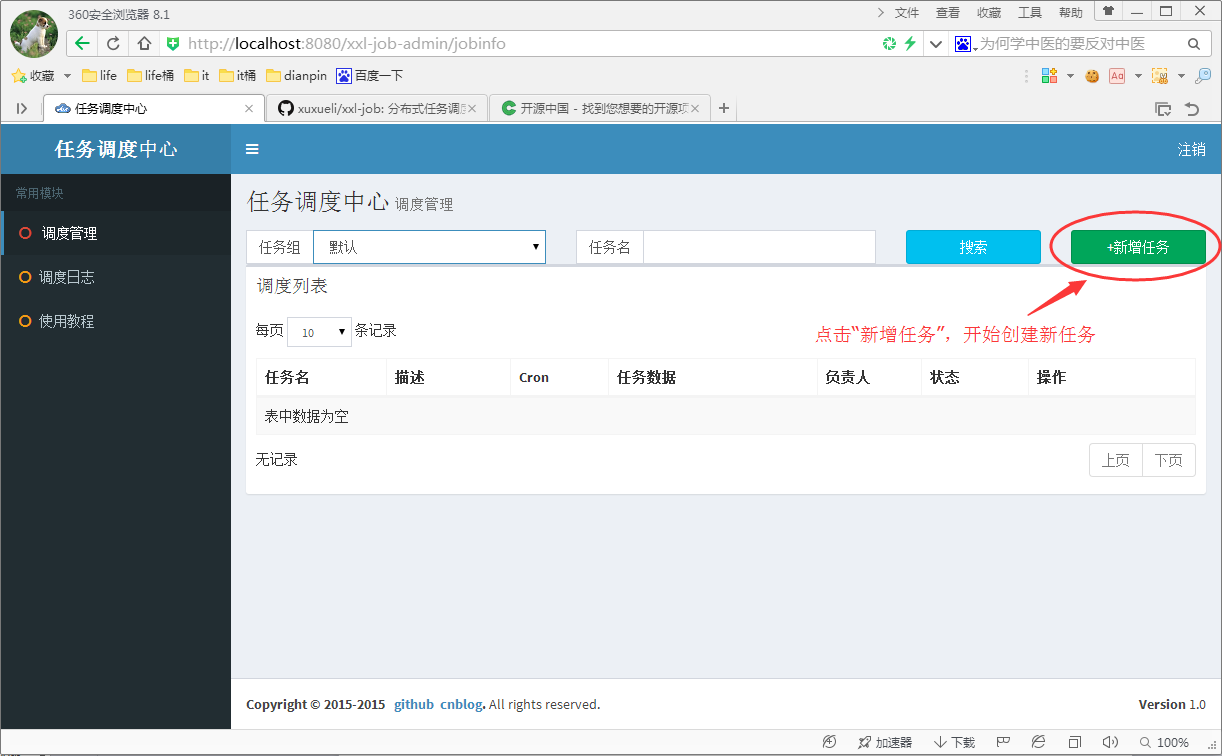
至此“执行器”项目已经部署结束。

## 2.5 开发第一个任务“Hello World”

本示例为新建一个“GLUE模式任务”（GLUE模式任务的执行代码支持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）。更多有关任务的详细配置，请查看“章节三：任务详解”。

**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）：**登陆调度中心，点击图2.5A所示新建任务按钮，新建示例任务。然后，参考图2.5B进行示例任务参数配置，点击保存。



（图2.5A：“调度中心”新建任务按钮）



（图2.5B：“调度中心”任务配置截图）

**步骤二（GLUE开发）：**请点击图2.5C中所示GLUE入口按钮，进入GLUE开发界面，如图2.5D。GLUE任务默认已经初始化了示例任务代码，即打印Hello World。

（GLUE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注入执行器里中的服务）



（图2.5C：“调度中心”管理管理界面，GLUE入口按钮）



（图2.5D：“调度中心”GLUE编辑界面）

**步骤三（触发执行）：**点击图2.5E所示“执行”按钮，可手动触发一次任务执行。

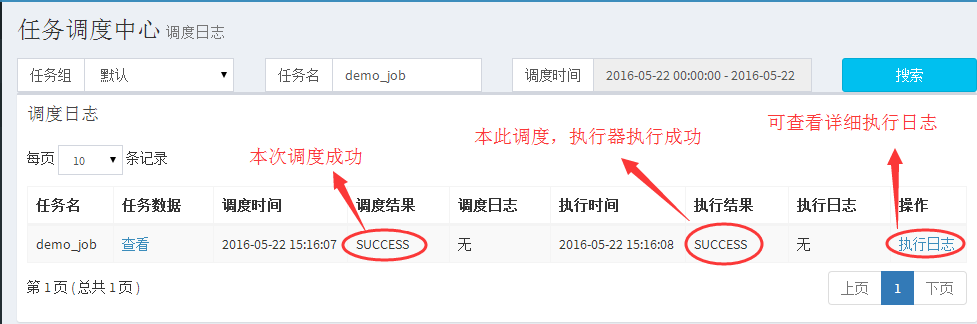


（图2.5E：“调度中心”管理管理界面，任务手动执行按钮接口）

**步骤四（查看日志）：**点击图2.5F所示“日志”按钮，可前往任务日志界面查看任务日志。在如图2.5G的任务日志界面中，可查看任务调度状态，执行器接收到调度请求后的执行状态，同时，点击如图2.5G中的“执行日志”按钮，可以查看本此调度在执行器端的完整执行日志，完整日志如图2.5H。



（图2.5F：“调度中心”管理管理界面，任务日志入口）



（图2.5G：“调度中心”管理管理界面，任务日志入口）



（图2.5H：“调度中心”管理管理界面，任务日志入口）

# 三 任务详解

## 4.1 BEAN模式任务

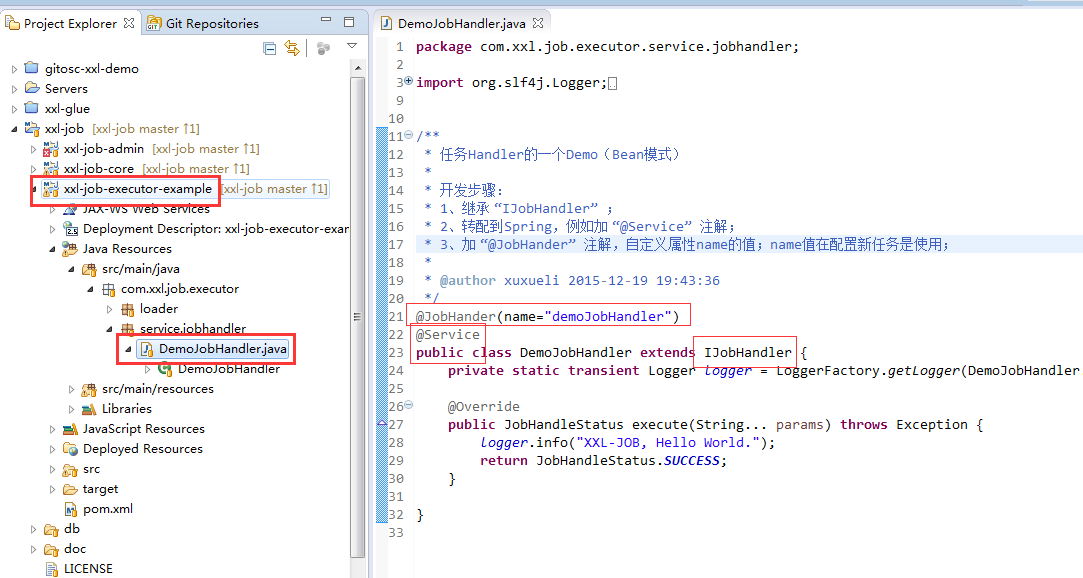
Bean模式任务：任务逻辑以JobHandler的形式存在于“执行器”所在项目中，开发流程如下：

**A：开发JobHandler**

如图4.1A所示，一个Bean模式的JobHandler需要配置以下三项：

1. 继承IJobHandler；
2. 被Spring容器扫描为Bean示例；
3. 添加@JobHander(name="demoJobHandler")注解，并定制handler名称，该名称是新建任务是JobHandler属性的值；

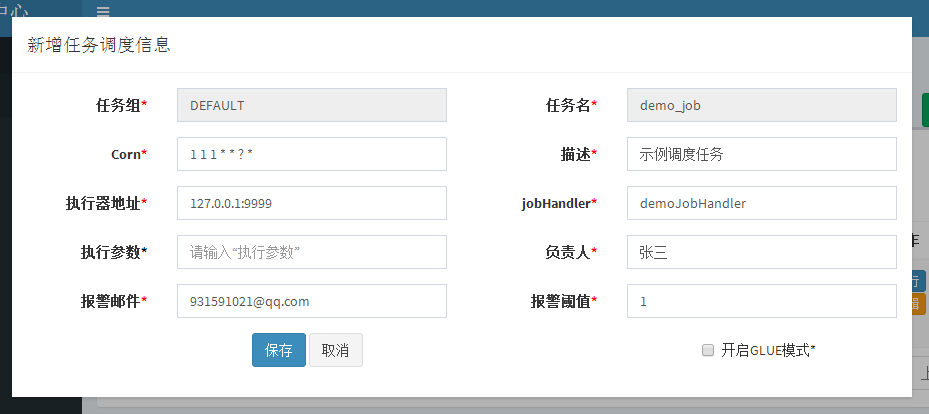
（例如xxl-job-executor-example项目中的DemoJobHandler）



（图4.1A：Bean模式任务的示例Handler）

**B：新建调度任务，并配置，参数如下**

|  |  |
| --- | --- |
| 任务组 | 任意选择 |
| 任务名 | 自定义，保证唯一即可 |
| Corn | 根据业务情况配置 |
| 描述 | 任意 |
| 执行器地址 | 执行器的地址，格式“IP：Port”，IP为“执行器”项目所在的机器的IP地址，因为本机部署所以为“127.0.0.1”，Port为“执行器”端口，如图2.4B所示，“xxl-job-client-demo”项目里的“执行器”配置的默认端口为9999，因为执行器地址为：127.0.0.1:9999 |
| JobHandler | Bean模式，任务代码在以JobHandler的形式部署在“执行器”所在的项目里。  此时应该填写对应任务JobHandler上注解@JobHander的name属性值，如图4.1A所示，示例任务name属性为demoJobHandler，因此此处填写demoJobHandler |
| 执行参数 | 任务执行的参数，多个入参都好分割，调用任务handler时会拆分成数组传入 |
| 负责人 | 任务负责人 |
| 报警邮件 | 任务失败时邮件通知的邮箱 |
| 报警阀值 | 任务失败次数超过该值才会发送报警邮件 |
| 开启GLUE模式 | 选择关闭状态 |

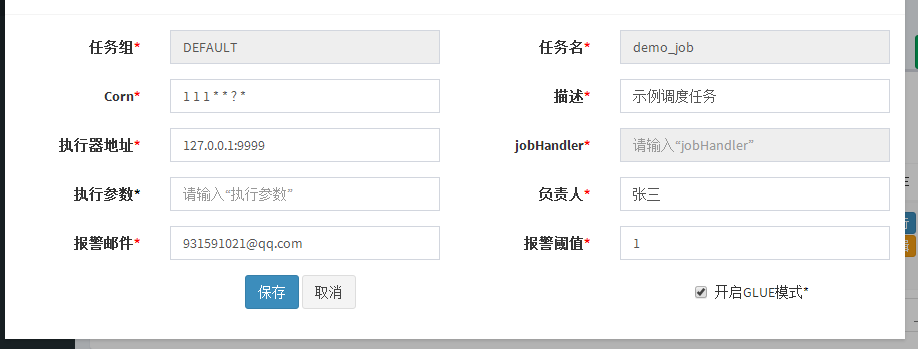


（图4.1B：Bean模式任务的示例配置）

## 4.3 GLUE模式任务

**A：新建调度任务，并配置，参数如下**

|  |  |
| --- | --- |
| 任务组 | 同Bean模式 |
| 任务名 | 同Bean模式 |
| Corn | 同Bean模式 |
| 描述 | 同Bean模式 |
| 执行器地址 | 同Bean模式 |
| JobHandler | GLUE 模式，业务代码保存在数据库中，并不需要“执行器”根据JobHandler去匹配任务Handler，因此此时该属性disable； |
| 执行参数 | 同Bean模式 |
| 负责人 | 同Bean模式 |
| 报警邮件 | 同Bean模式 |
| 报警阀值 | 同Bean模式 |
| 开启GLUE模式 | 选择开启状态； |



（图4.2A：Glue模式任务的示例配置）

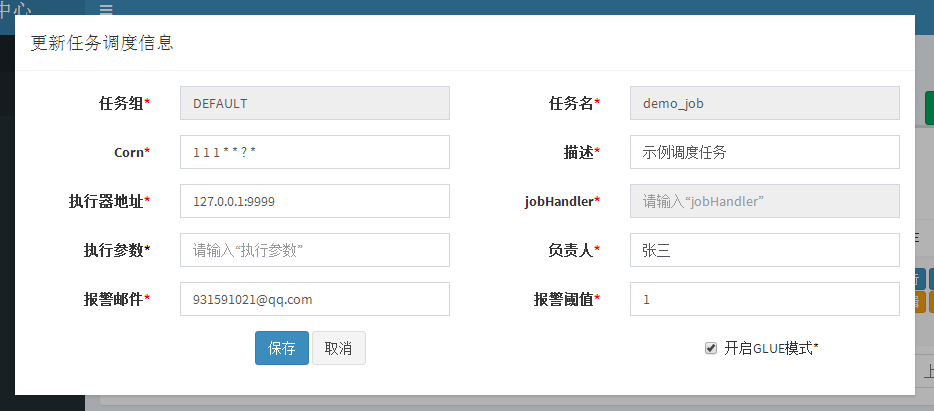
# 四 任务管理

## 5.1 编辑任务信息

在调度管理界面，如图5.1A所示，点击“编辑”按钮，弹出更新任务界面，界面如图5.1B所示。修改属性值，保存后实时生效。



（图5.1A：调度管理列表，“编辑”按钮）



（图5.1B：调度管理列表，更新任务界面）

## 5.2 编辑GLUE代码

表操作仅针对GLUE任务，如图5.2A所示在调度管理界面点击“GLUE”按钮，进入GLUE开发界面，界面如图5.2B所示；在GLUE编辑器中可对GLUE代码进行开发；（当然也可以在IDE中开发完成后，复制粘贴到编辑中）

**版本回溯功能：**如图5.2B，选择下拉框“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；



（图5.2A：GLUE入口按钮）



（图5.2B：GLUE开发界面）

## 5.3 恢复/暂停

可对调度任务进行暂停和回复操作；



（图5.3：暂停/回复按钮）

## 5.4 手动触发一次调度

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。



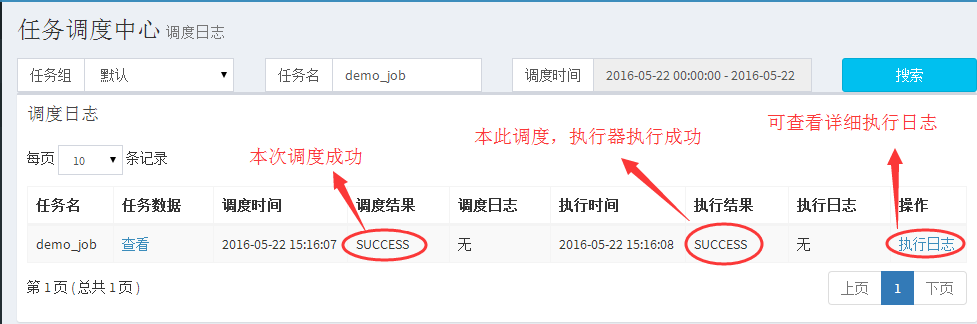
（图5.3：执行按钮）

## 5.5 查看日志

如图5.5A所示，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（见图5.5B）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（如图5.5C）；



（图5.5A：任务日志入口）



（图5.5B：执行日志）



（图5.5C：完整的执行日志）

## 5.6 删除任务

点击删除按钮，可以删除对应任务。



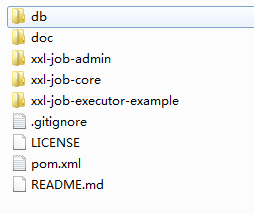
## 5.7 终止运行中的任务

进入调度日志界面，如图5.7所示可查看目前正在运行的任务，如果任务正在运行中，右侧将会显示“终止任务”按钮，点击按钮将会终止任务执行。

 （图5.7：完整的执行日志）

# 五 总体设计

## 5.1 源码目录介绍



（图5.1：源码目录截图）

|  |  |
| --- | --- |
| 目录 | 介绍 |
| /doc | 用户手册 |
| /db | “调度数据库”建表脚本 |
| /xxl-job-admin | “调度中心”Wed项目源码 |
| /xxl-job-core | “执行器”Jar依赖 |
| /xxl-job-executor-example | “执行器”Wed项目源码 |

其中，Wed项目“xxl-job-executor-example”是Demo执行器项目，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

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\_），如图3.2A所示。

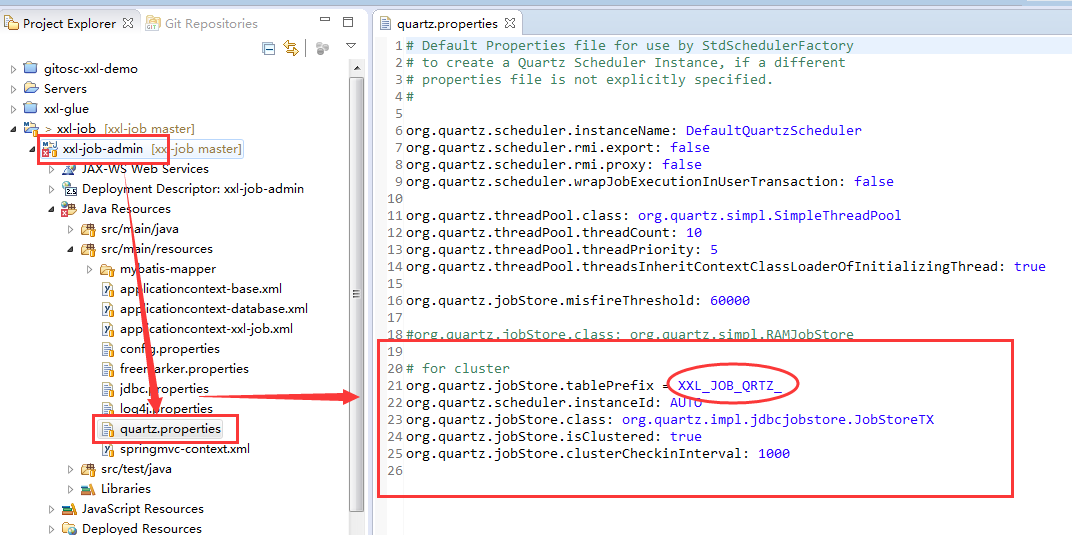
然后，在此基础上新增了三张扩展表，如下：

xxl\_job\_qrtz\_trigger\_info：

xxl\_job\_qrtz\_trigger\_log：

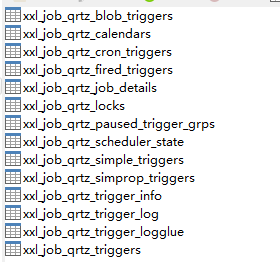
xxl\_job\_qrtz\_trigger\_logglue：

因此，XXL-JOB调度数据库共计用于14张数据库表，详细介绍如图5.2B所示。



（图5.2A：XXL-JOB数据库，公共前缀配置，截图）

**表介绍：**XXL-JOB拥有14张表，



（图3.2B：调度数据库，表列表截图）

|  |  |
| --- | --- |
| 表明 | 介绍 |
| xxl\_job\_qrtz\_blob\_triggers | Quartz表 |
| xxl\_job\_qrtz\_calendars | Quartz表 |
| xxl\_job\_qrtz\_cron\_triggers | Quartz表 |
| xxl\_job\_qrtz\_fired\_triggers | Quartz表 |
| xxl\_job\_qrtz\_job\_details | Quartz表 |
| xxl\_job\_qrtz\_locks | Quartz表 |
| xxl\_job\_qrtz\_paused\_trigger\_grps | Quartz表 |
| xxl\_job\_qrtz\_scheduler\_state | Quartz表 |
| xxl\_job\_qrtz\_simple\_triggers | Quartz表 |
| xxl\_job\_qrtz\_simprop\_triggers | Quartz表 |
| 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； |
| xxl\_job\_qrtz\_triggers | Quartz表 |

## 5.3 架构设计

系统主要由以下三个基础模块组成：

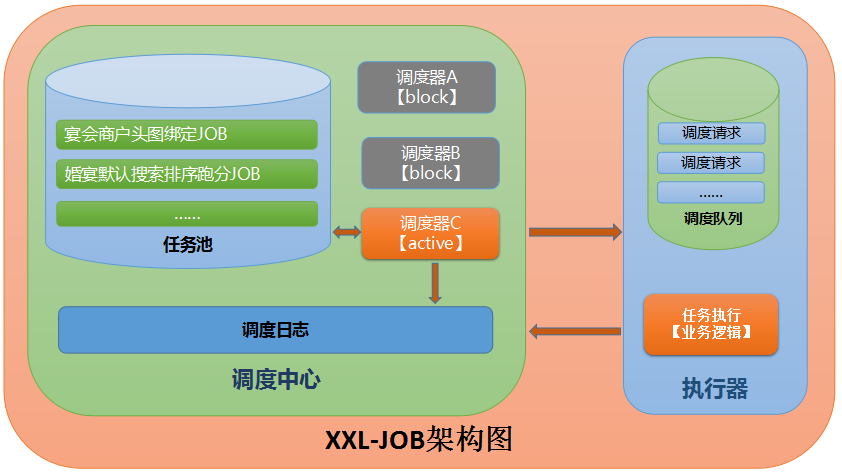
**调度模块（调度中心xxl-job-admin）：**负责管理调度信息，按照调度配置发出调度请求，自身不承担业务代码。调度系统与任务解耦，提高了系统可用性和稳定性，同时调度系统性能不再受限于任务模块；

支持可视化、简单且动态的维管理调度信息，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。

**执行模块（执行器xxl-job-client-demo）：**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，开发和维护更加简单和高效；

负责接收“调度中心”的调度请求（运行/终止/日志）。

**通讯模块：**负责调度模块和任务模块之间的信息通讯；

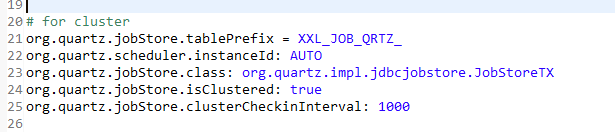


## 6.4 调度模块

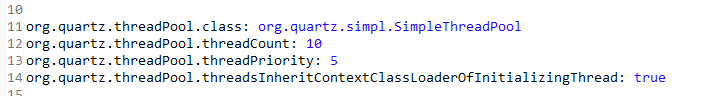
**A：RemoteHttpJobBean：**常规Quartz的开发，任务逻辑一般维护在QuartzJobBean中，耦合很严重。XXL-JOB中“调度模块”和“任务模块”完全解耦，调度模块中的所有调度任务使用同一个QuartzJobBean，即RemoteHttpJobBean。不同的调度任务将各自参数维护在各自扩展表数据中，当触发RemoteHttpJobBean执行时，将会解析不同的任务参数发起远程调用，调用各自的远程执行器服务。

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。

**B：集群分布式**：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。

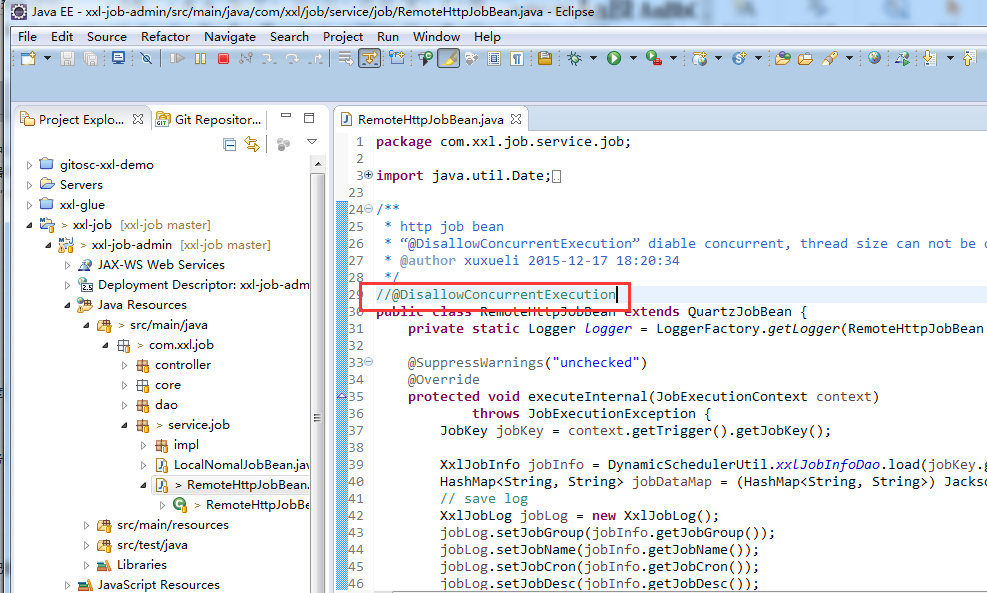


**C：调度线程池：**默认线程池中线程的数量为10个，避免单线程因阻塞而引起任务调度延迟。



**D：@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。



**E：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，单位毫秒：



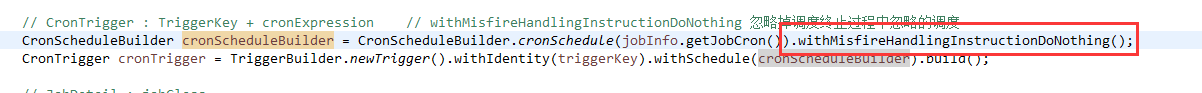
Misfire规则：

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；

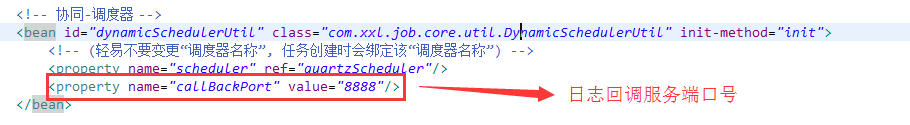
XXL-JOB默认misfire规则为：withMisfireHandlingInstructionDoNothing



**F：日志回调服务：**

调度模块的“调度中心”作为Web服务单独部署，除此之外，内部嵌入jetty服务器提供日志回调服务。

“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”，如下图所示。



## 6.5 执行模块

**A：Bean模式任务：**

开发步骤：

1. 开发一个继承“IJobHandler”的实现类，任务逻辑写在execute方法中；
2. 将任务类装配到Spring中，例如加 “@Service” 注解；
3. 在任务类上加 “@JobHander” 注解，定义注解的name属性值；

（步骤3中定义的name属性值，在配置Bean模式任务是会用到）

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(name="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。

**B：GLUE模式任务：**

开发步骤：GLUE模式任务的开发步骤比较简单，进入GLUE编辑界面，在Demo任务代码的execute方法中写入任务逻辑即可；可使用@Resource或@Autoward注解注入“执行器”项目中的Spring服务；

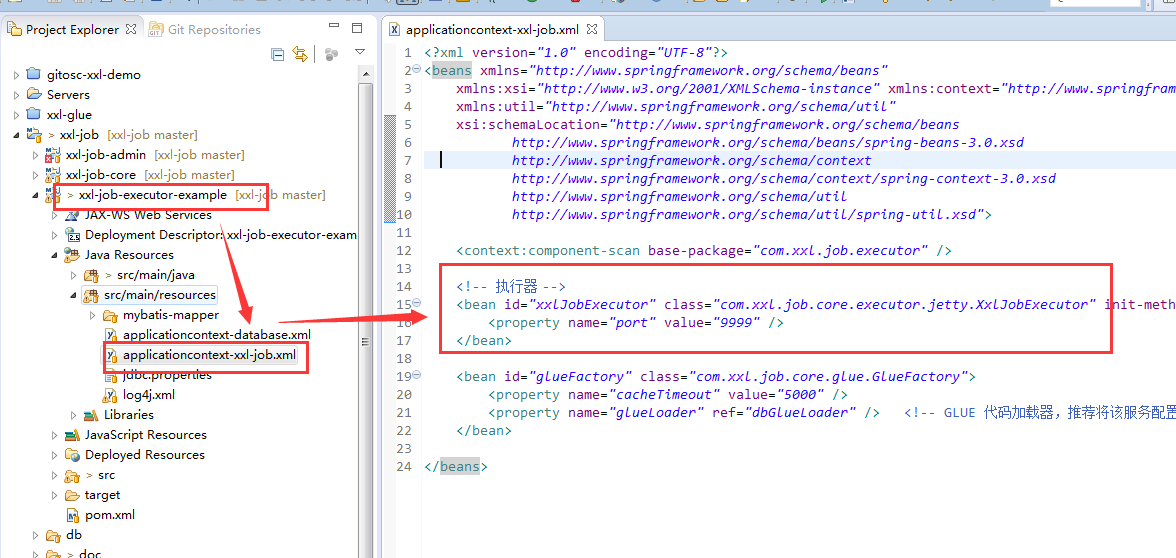
原理：每个Glue任务的代码，实际上是“一个继承自“IJobHandler”的实现类的类代码”，“执行器”接收到“调度中心”的调度请求时，会通过Groovy类加载器加载此代码，实例化成Java对象，同时注入此代码中声明的Spring服务（请确保Glue代码中的服务和类引用在“执行器”项目中存在），然后调用该对象的execute方法，执行任务逻辑。

**C：执行器：**

执行器实际上是一个内嵌的Jetty服务器，默认端口9999。

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name属性为key管理起来。

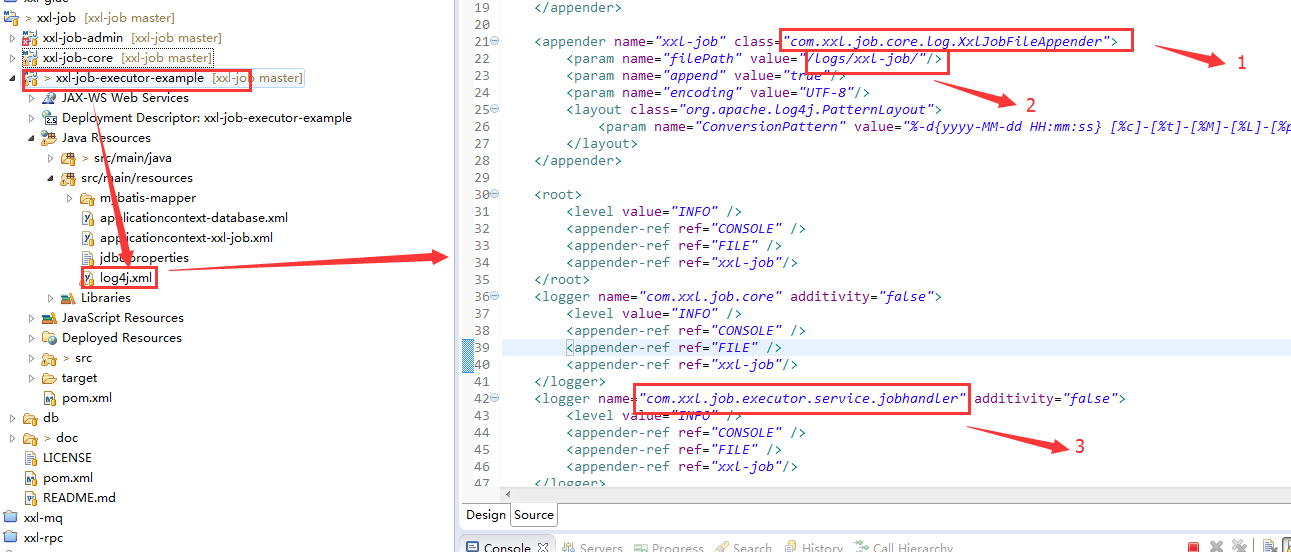
“执行器”接收到“调度中心”的调度请求时，如果任务类型为“Bean模式”，将会匹配Spring容器中的“Bean模式任务”，然后调用其execute方法，执行任务逻辑。如果任务类型为“GLUE模式”，将会加载GLue代码，实例化Java对象，注入依赖的Spring服务，然后调用execute方法，执行任务逻辑。



**D：任务日志：**

XXL-JOB会为每次调度请求生成一个单独的日志文件，通过重写LOG4J的Appender实现如图6.6D1的数据1处），“调度中心”查看执行日志时将会加载对应的日志文件。

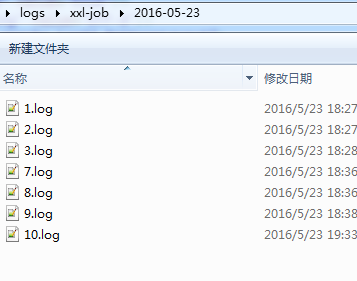
需要注意的是，“执行器”中日志Appender上配置的包名（如图6.6D1的数据3处），需要覆盖到所有任务（Bean模式 + GLUE模式）的包名，否则覆盖不到的任务将不会生成日志文件。



（图6.6D1：“执行器”任务日志配置）

单独日志文件存放的位置可在“执行器”的log.xml文件进行自定义，如图6.6D1中的数字2的位置所示，默认位置为项目磁盘根目录下“/logs/xxl-job/”；

目录格式为：/logs/xxl-job/ “格式化日期”/“数据库调度日志记录的主键ID . log”，如图6.6D2所示。



（图6.6D2：“执行器”任务日志目录）

## 6.6 通讯模块

一次完整的任务调度流程如下：

1. “调度中心”向“执行器”发送http调度请求。如图2.4B所示。

（“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）

1. “执行器”执行任务逻辑；
2. “执行器”http回调“调度中心”调度结果；如图2.3C所示。

（“调度中心”中接收回调的服务，实际上是一台内嵌jetty服务器，默认端口8888）

# 七 其他

## 7.1 接入登记

更多接入公司，欢迎在<https://github.com/xuxueli/xxl-job/issues/1> 登记。

## 7.2 报告问题

XXL-JOB托管在Github上，如有问题可在[ISSUES](https://github.com/xuxueli/xxl-job/issues)上提问，也可以加入技术交流群(仅作技术交流)：367260654 。