

XXL-JOB 1.3 用户手册

一 简介.....	2
1.1 背景.....	2
1.2 特点.....	2
1.3 发展.....	2
1.4 下载.....	3
1.5 环境.....	3
二 快速入门.....	4
2.1 初始化“调度数据库”.....	4
2.2 编译源码.....	5
2.3 配置部署“调度中心”.....	5
2.4 配置部署“执行器项目”.....	7
2.5 开发第一个任务“Hello World”.....	9
三 任务详解.....	12
4.1 BEAN 模式任务.....	12
4.3 GLUE 模式任务.....	13
四 任务管理.....	15
5.1 编辑任务信息.....	15
5.2 编辑 GLUE 代码.....	15
5.3 恢复/暂停.....	16
5.4 手动触发一次调度.....	16
5.5 查看日志.....	16
5.6 删除任务.....	17
5.7 终止运行中的任务.....	17
五 总体设计.....	19
5.1 源码目录介绍.....	19
5.2 “调度数据库”配置.....	19
5.2 架构设计.....	21
6.3 调度模块.....	22
6.4 任务模块.....	24
6.5 通讯模块.....	24
七 其他.....	25
7.1 接入登记.....	25
7.2 报告问题.....	25

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。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，动态暂停/恢复任务，即时生效；
- 3、服务 HA：任务信息持久化到 mysql 中，Job 服务天然支持集群，保证服务 HA；
- 4、任务 HA：某台 Job 服务挂掉，任务会平滑分配给其他的某一台存活服务，即使所有服务挂掉，重启时或补偿执行丢失任务；
- 5、一个任务只会在其中一台服务器上执行；
- 6、任务串行执行；
- 7、支持任务执行日志；
- 8、支持自定义参数；
- 9、支持任务失败次数超阈值邮件报警；
- 10、支持在线查看，执行器详细日志；
- 11、支持远程任务执行终止；
- 12、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。

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

技术交流群(仅作技术交流)：367260654

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。

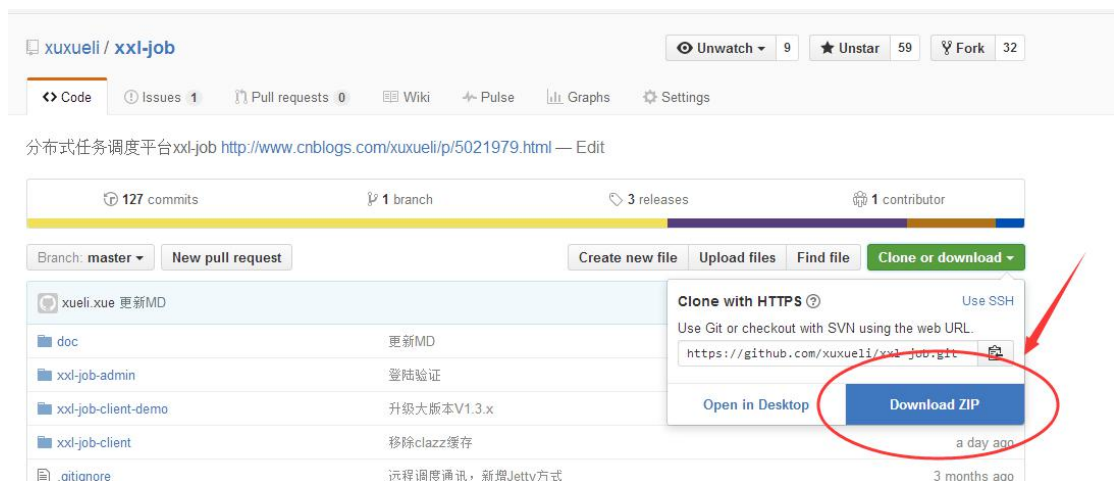
至今, XXL-JOB 已接入多家公司的线上产品线, 场景如电商业务, O2O 业务和大数据作业等等, 欢迎大家使用, XXL-JOB 也将拥抱变化, 持续发展。

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

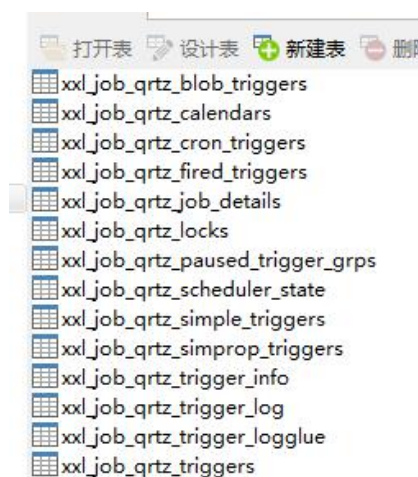
2.1 初始化“调度数据库”

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\doc\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，



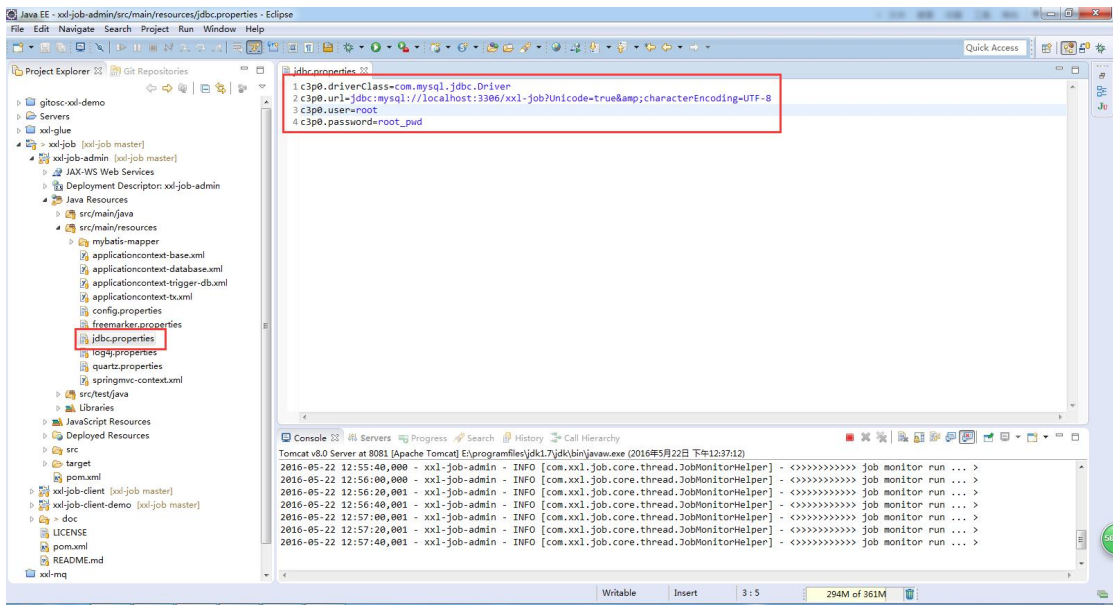
(图 2.2: 项目源码导入 Eclipse 截图)

2.3 配置部署“调度中心”

“调度中心”项目：xxl-job-admin

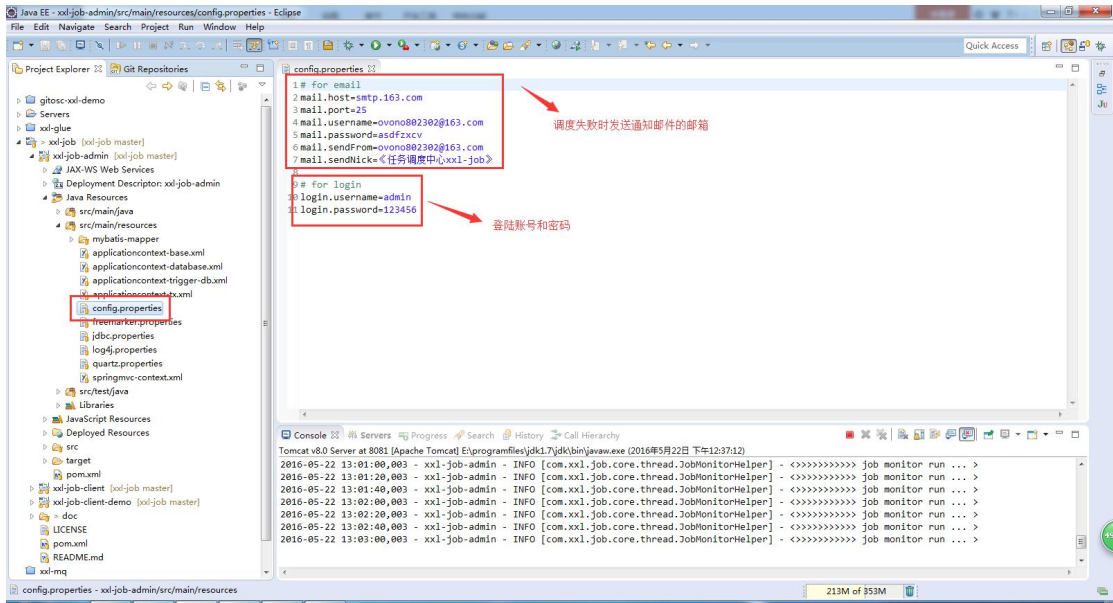
作用：统一管理任务调度平台上调度任务，负责触发调度执行。

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。



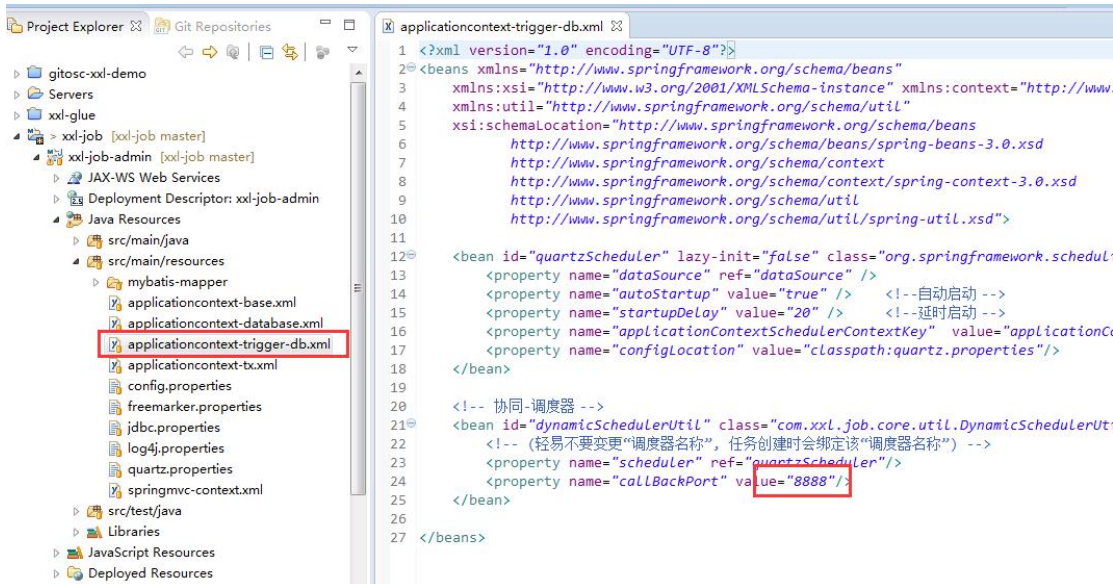
(图 2.2A: 数据库 JDBC 配置截图)

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。



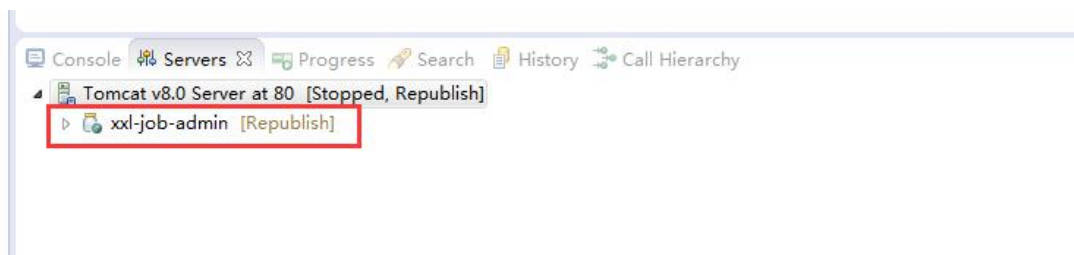
(图 2.2B: 系统配置截图)

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)



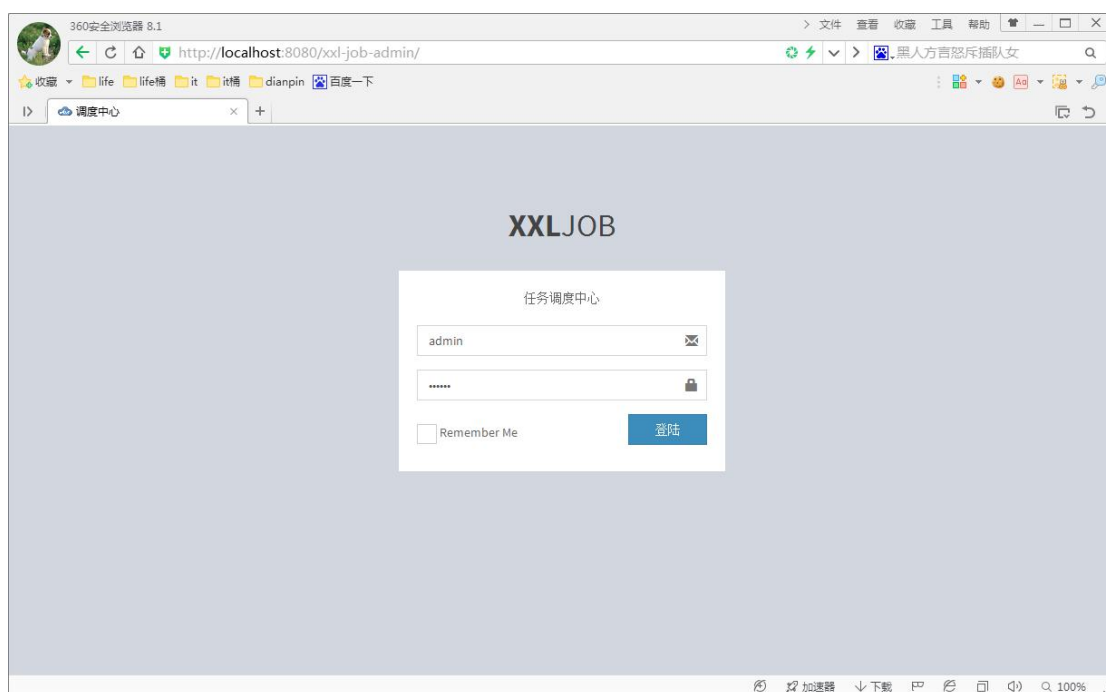
(图 2.2C)

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。



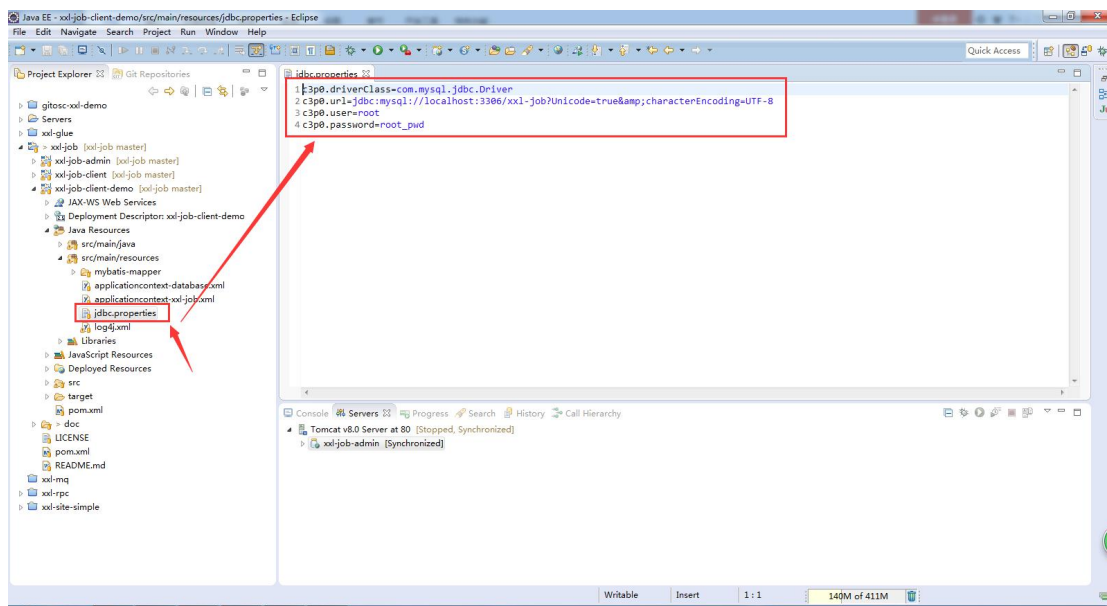
(图 2.2E: “调度中心” 登陆界面)

2.4 配置部署“执行器项目”

“执行器”项目: `xxl-job-client-demo`

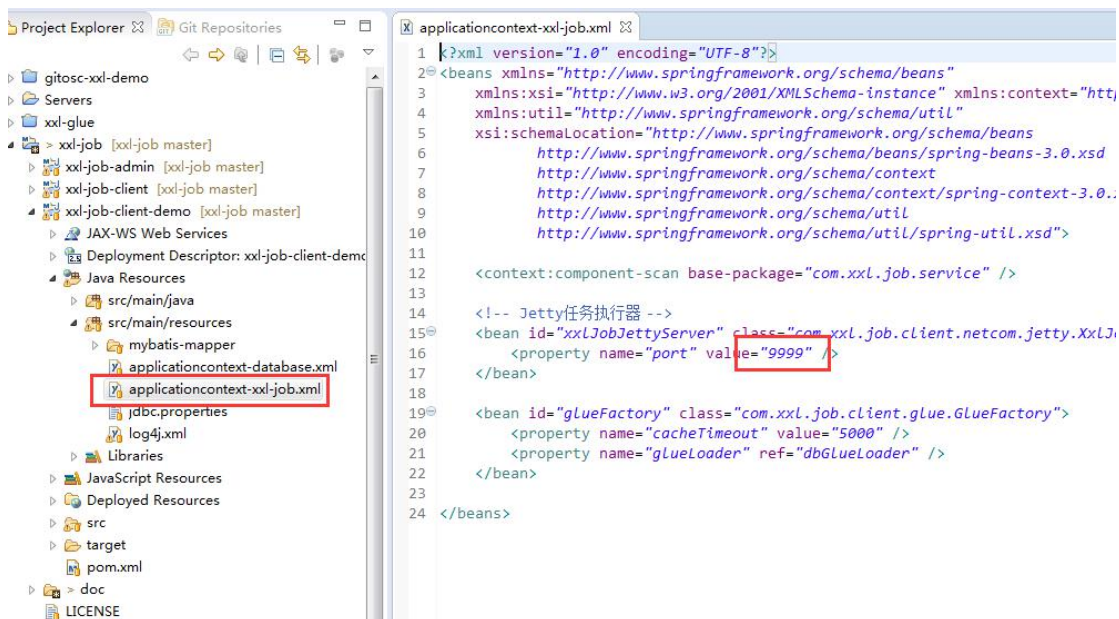
作用: 负责接收“调度中心”的调度并执行;

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。

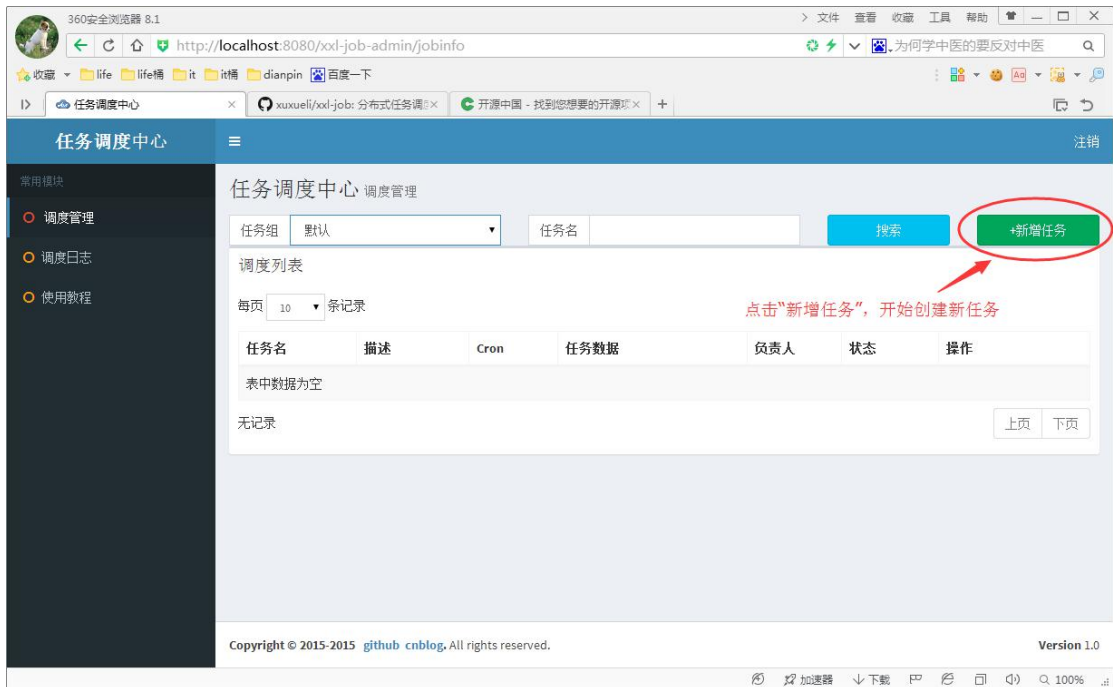
至此“执行器”项目已经部署结束。

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）



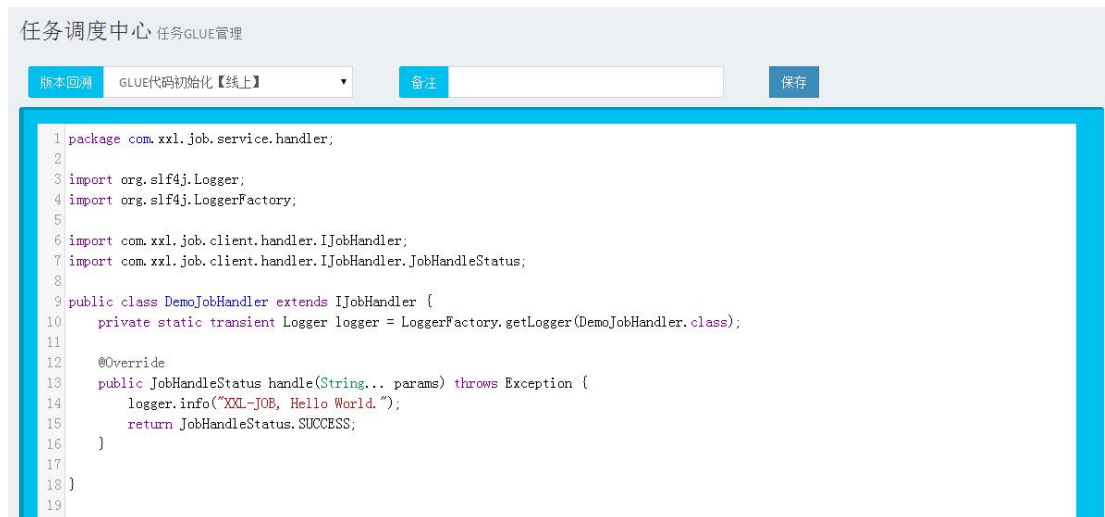
（图 2.5B：“调度中心”任务配置截图）

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。

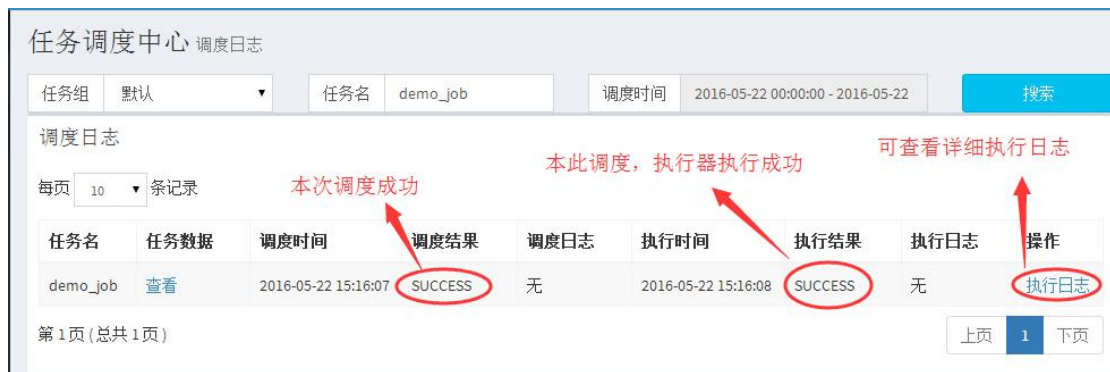


（图 2.5E：“调度中心”管理管理界面，任务手动执行按钮接口）

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)

```

2016-05-22 15:16:07 [com.xx1.job.client.handler.HandlerThread-[Thread-14]-[run]-[81]-[INFO]] >>>>>>> xxl-job handle start.
2016-05-22 15:16:07 [com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource-[Thread-14]-[getPoolManager]-[462]-[INFO]] Initializing c3p0 pool... com.mchange.v2.c
2016-05-22 15:16:08 [com.xx1.job.client.glue.ClrFactory-[Thread-14]-[readInstance]-[153]-[INFO]] >>>>>>> read glue, fresh instance, cacheInstanceKey:DEFAULT
2016-05-22 15:16:08 [com.xx1.job.service.handler.DemoJobHandler-[Thread-14]-[call]-[?] -[INFO]] XXL-JOB, Hello World.
2016-05-22 15:16:08 [com.xx1.job.client.handler.HandlerThread-[Thread-14]-[run]-[89]-[INFO]] >>>>>>> xxl-job handle end, handlerParams:[], _status:SUCCESS,
2016-05-22 15:16:08 [com.xx1.job.client.handler.HandlerThread-[Thread-14]-[run]-[98]-[INFO]] >>>>>>> xxl-job callback start.
2016-05-22 15:16:08 [com.xx1.job.client.handler.HandlerThread-[Thread-14]-[run]-[104]-[INFO]] >>>>>>> xxl-job callback end, params:{status=SUCCESS, trigger_

```

(图 2.5H: “调度中心”管理管理界面，任务日志入口)

三 任务详解

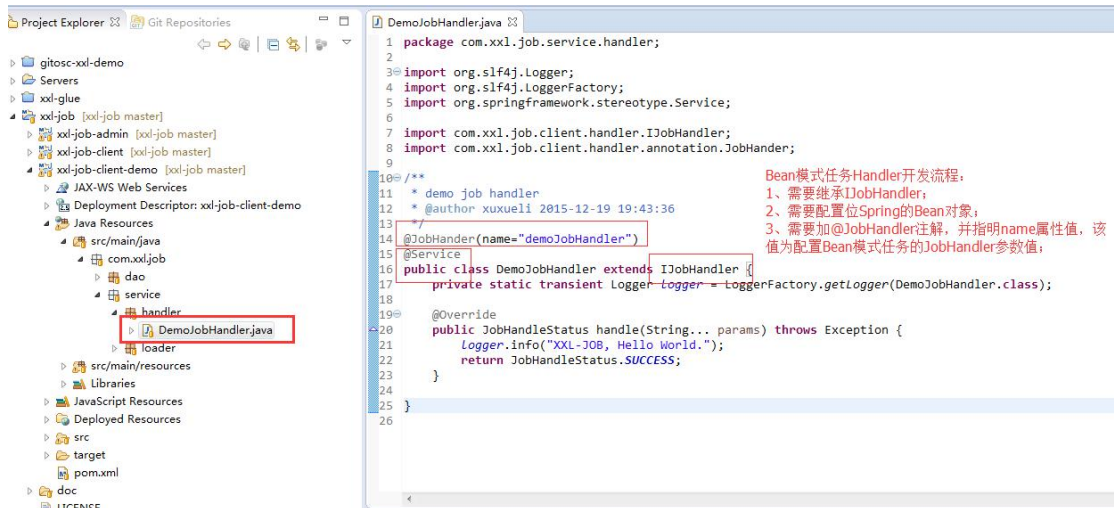
4.1 BEAN 模式任务

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@JobHandler(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-client-demo 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 @JobHandler 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
报警阈值	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
报警阈值	同 Bean 模式
开启 GLUE 模式	选择开启状态;

任务组*	DEFAULT	任务名*	demo_job
Corn*	1 1 1 * * ? *	描述*	示例调度任务
执行器地址*	127.0.0.1:9999	jobHandler*	请输入“jobHandler”
执行参数*	请输入“执行参数”	负责人*	张三
报警邮件*	931591021@qq.com	报警阈值*	1
<input type="button" value="保存"/> <input type="button" value="取消"/>		<input checked="" type="checkbox"/> 开启GLUE模式*	

(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。

任务名	描述	Cron	任务数据	负责人	状态	操作
demo_job	示例调度任务	1111**?*	执行器： 执行参数： 执行器地址：127.0.0.1:9999	张三	ONORMAL	执行 暂停 日志 编辑 GLUE 删除

第 1 页 (总共 1 页) 上页 1 下页

点击编辑按钮，进入任务编辑界面

(图 5.1A: 调度管理列表，“编辑”按钮)

更新任务调度信息

<p>任务组* <input type="text" value="DEFAULT"/></p> <p>Corn* <input type="text" value="1111**?*"/></p> <p>执行器地址* <input type="text" value="127.0.0.1:9999"/></p> <p>执行参数* <input type="text" value="请输入“执行参数”"/></p> <p>报警邮件* <input type="text" value="931591021@qq.com"/></p>	<p>任务名* <input type="text" value="demo_job"/></p> <p>描述* <input type="text" value="示例调度任务"/></p> <p>jobHandler* <input type="text" value="请输入“jobHandler”"/></p> <p>负责人* <input type="text" value="张三"/></p> <p>报警阈值* <input type="text" value="1"/></p>
---	---

开启GLUE模式*

(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）

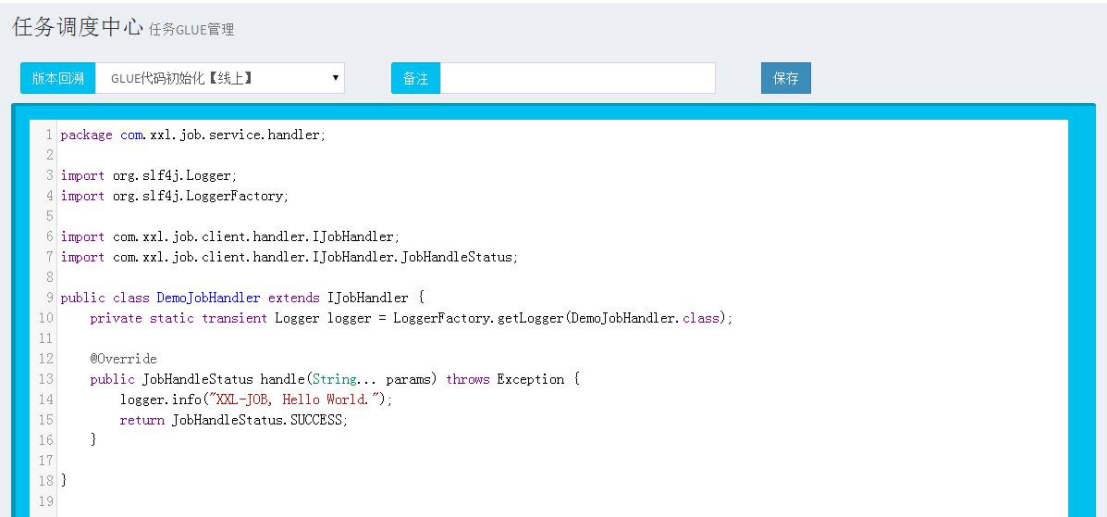
版本回溯功能：如图 5.2B，选择下拉框“版本回溯”，会列出该 GLUE 的更新历史，选择相应版本即可显示该版本代码，保存后 GLUE 代码即回退到对应的历史版本；

任务名	描述	Cron	任务数据	负责人	状态	操作
demo_job	示例调度任务	1111**?*	执行器： 执行参数： 执行器地址：127.0.0.1:9999	张三	ONORMAL	执行 暂停 日志 编辑 GLUE 删除

第 1 页 (总共 1 页) 上页 1 下页

点击“GLUE”按钮，进入GLUE开发界面

(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。



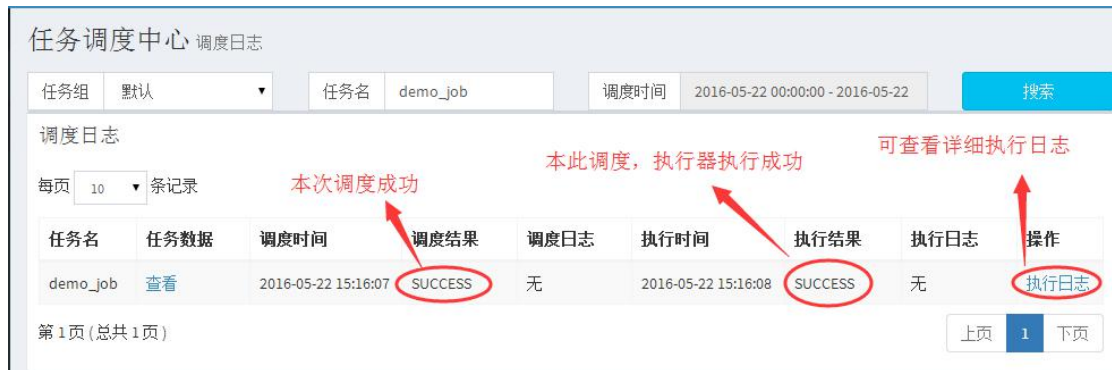
(图 5.3: 执行按钮)

5.5 查看日志

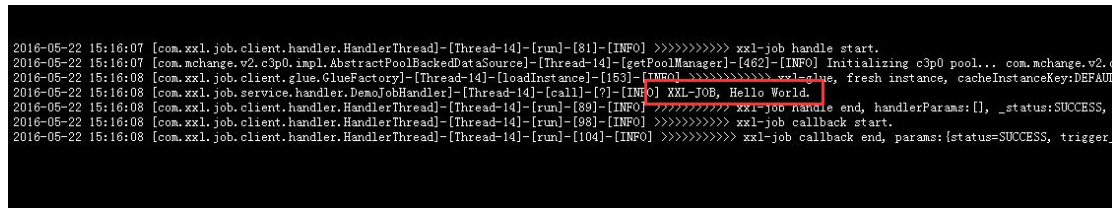
如图 5.5A 所示，点击“日志”按钮，可以查看任务历史调度日志。在历史调入日志界面（见图 5.5B）可查看每次任务调度的调度结果、执行结果等，点击执行日志按钮可查看执行器完整日志（如图 5.5C）；



(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。

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

技术交流群(仅作技术交流): 367260654

调度日志

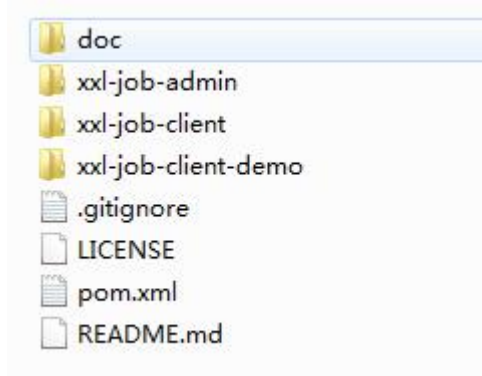
每页 条记录 支持手动终止运行中的任务

任务名	任务数据	调度时间	调度结果	调度日志	执行时间	执行结果	执行日志	操作
demo_job	查看	2016-05-22 21:34:46	SUCCESS	无	2016-05-22 21:34:48			终止任务

(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
/doc/db	“调度数据库”建表脚本
/xxl-job-admin	“调度中心”Wed 项目源码
/xxl-job-client	“执行器”Jar 依赖
/xxl-job-client-demo	“执行器”示例 Wed 项目源码。

其中，Wed 项目“xxl-job-client-demo”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

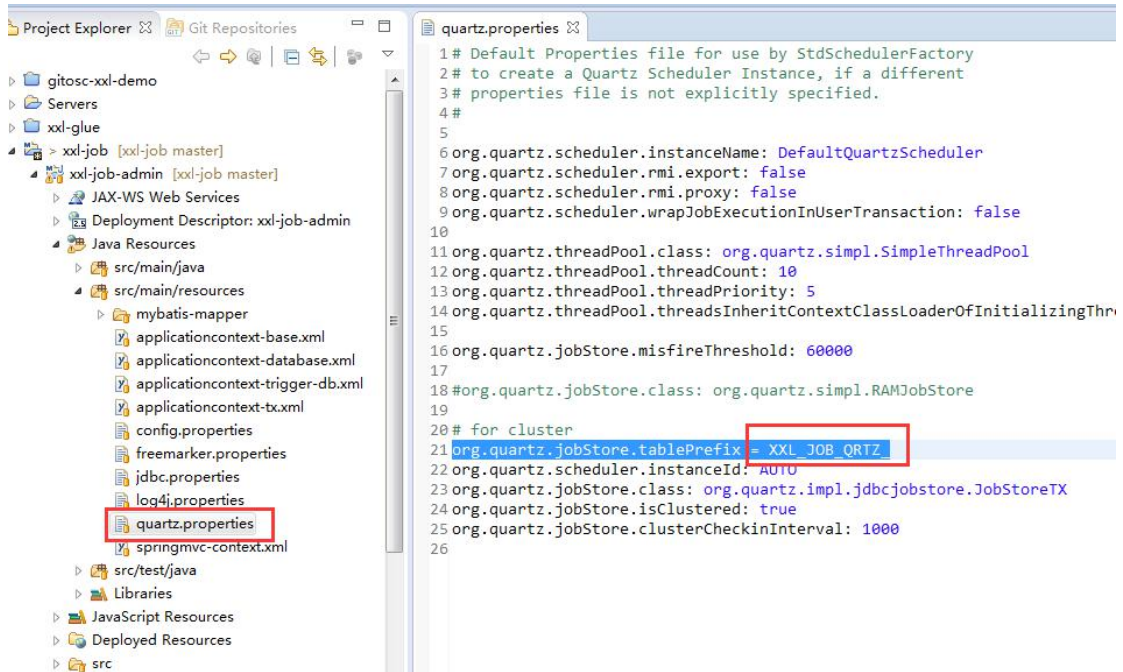
XXL-JOB 调度模块基于 Quartz 集群实现，其“调度数据库”是在 Quartz 的 11 张集群 mysql 表基础上扩展而成。

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：

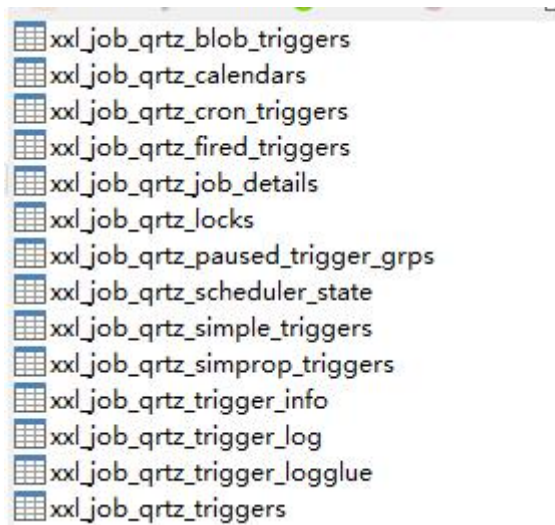
```
xxl_job_qrtz_trigger_info:
xxl_job_qrtz_trigger_log:
xxl_job_qrtz_trigger_logglue:
```

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

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

技术交流群(仅作技术交流): 367260654

xxl_job_quartz_simple_triggers	Quartz 表
xxl_job_quartz_simprop_triggers	Quartz 表
xxl_job_quartz_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_quartz_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_quartz_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_quartz_triggers	Quartz 表

5.2 架构设计

系统主要由以下三个基础模块组成：

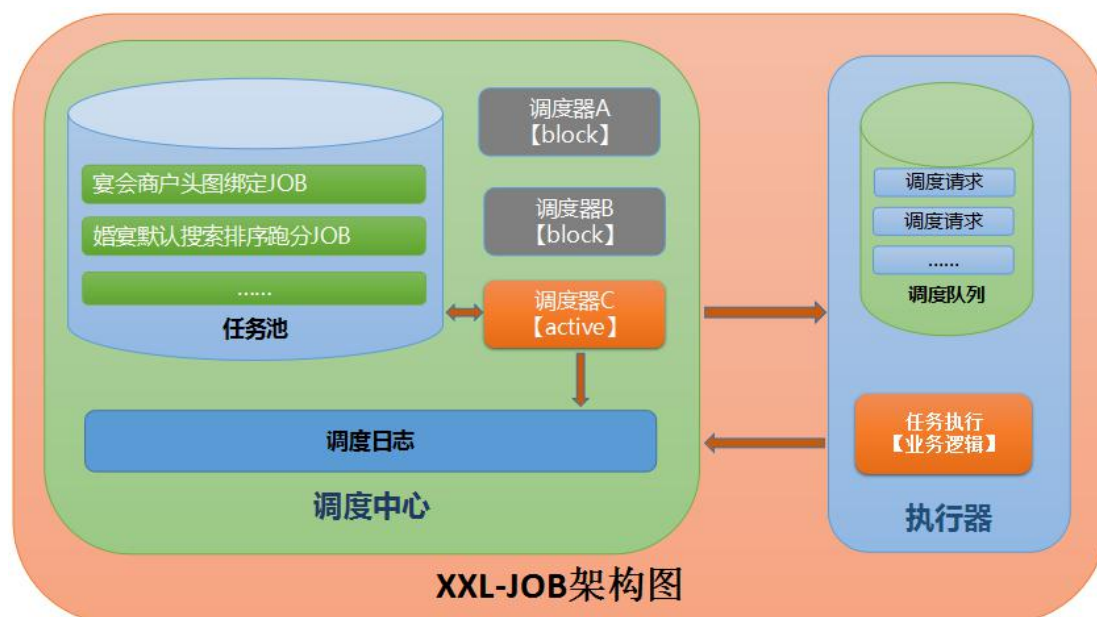
调度模块（调度中心 xxl-job-admin）：负责管理调度信息，按照调度配置发出调度请求，自身不承担业务代码。调度系统与任务解耦，提高了系统可用性和稳定性，同时调度系统性能不再受限于任务模块；

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。

任务模块（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；



6.3 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。

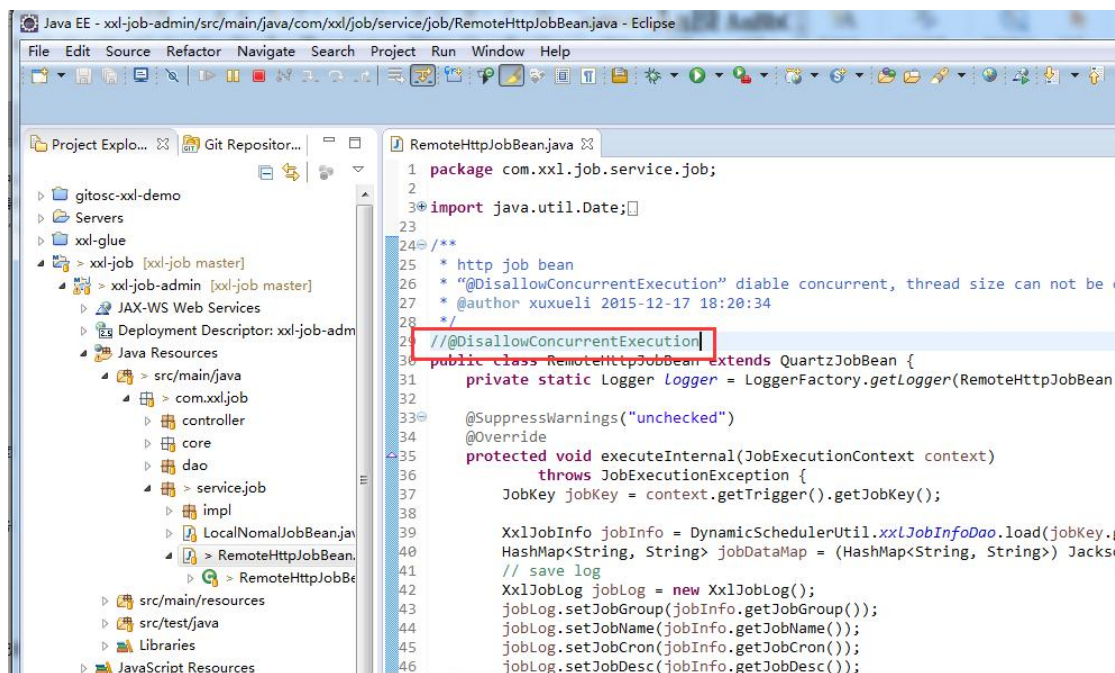
```
19|
20# for cluster
21 org.quartz.jobStore.tablePrefix = XXL_JOB_QRTZ_
22 org.quartz.scheduler.instanceId: AUTO
23 org.quartz.jobStore.class: org.quartz.impl.jdbcjobstore.JobStoreTX
24 org.quartz.jobStore.isClustered: true
25 org.quartz.jobStore.clusterCheckinInterval: 1000
26
```

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。

```
10
11 org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool
12 org.quartz.threadPool.threadCount: 10
13 org.quartz.threadPool.threadPriority: 5
14 org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true
15
```

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 的阈值配置，单位毫秒：

```

15
16 org.quartz.jobStore.misfireThreshold: 60000
17

```

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

```

// CronTrigger : TriggerKey + cronExpression // withMisfireHandlingInstructionDoNothing 忽略该调度终止过程中忽略的调度
CronScheduleBuilder cronScheduleBuilder = CronScheduleBuilder.cronSchedule(jobInfo.getJobCron()).withMisfireHandlingInstructionDoNothing();
CronTrigger cronTrigger = TriggerBuilder.newTrigger().withIdentity(triggerKey).withSchedule(cronScheduleBuilder).build();


```

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。

```
<!-- 协同-调度器 -->  
<bean id="dynamicSchedulerUtil" class="com.xxl.job.core.util.DynamicSchedulerUtil" init-method="init">  
  <!-- (轻易不要变更“调度器名称”，任务创建时会绑定该“调度器名称”) -->  
  <property name="scheduler" ref="quartzScheduler"/>  
  <property name="callBackPort" value="8888"/>  
</bean>
```



6.4 任务模块

任务开发: 请参考章节三;

执行器详解:

任务日志:

6.5 通讯模块

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](#) 上提问，也可以加入技术交流群(仅作技术交流)：367260654 。