ALIBABA CLOUD

阿里云

分布式任务调度 SchedulerX

常见问题

文档版本：20210408

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.从分布式任务调度1.0迁移到2.0

自2019年6月30日起，分布式任务调度SchedulerX 1.0（DTS）不再提供运维服务，提供服务的ECS实例也会裁撤，待所有用户下线后会完全下线，后续任务调度服务将由SchedulerX 2.0提供。本文介绍如何从SchedulerX 1.0迁移到SchedulerX 2.0。

迁移说明

1. 在EDAS 组件概览页面开通分布式任务调度2.0。
2. 进入分布式任务调度 2.0 应用管理页面，创建分组，groupId要和分布式任务管理1.0保持一致。
3. 使用迁移工具进行迁移。
4. 升级JAR包，重新发布应用。
5. 在EDAS控制台通过分布式任务调度2.0验证接入成功。

注释事项

- 目前只支持1000个任务以内的单个分组迁移到2.0，如果单分组任务数超过1000，请联系SchedulerX支持人员。
- 一次性任务（指定具体某时某分执行的任务，工具会判断并打印出jobId）不支持迁移，有特殊需求的联系SchedulerX支持人员。
- 原有报警不会迁移，如果任务有报警需求的需要手动订正。
- 所有的秒级任务（如 `0/10****?` ）迁移到2.0会变为second_delay任务，同时delay时间为10秒。
- **同一个分组多次执行迁移一定要在同一台机器上执行**，否则服务端会重复新增，所有成功迁移任务ID存储在客户端执行机器上。
- 2.0部分地域（Region）不支持经典网络的机器。

升级JAR包的方案

分布式任务调度SchedulerX 从1.0迁移到2.0有重建和兼容两种方案。

方案一：重建（推荐）

使用迁移工具把SchedulerX 1.0的Job全量导入到2.0，然后使用SchedulerX 2.0的官方正式包（不兼容1.0的接口）修改代码。

- 优势：
  - 使用2.0的接口和编程模型，可以享受到2.0的功能，例如可以直接在前端看到运行日志。
  - 2.0会不断迭代，每次升级都会增加新的功能并进行bug fix。（兼容版本客户端只会发布一个版本，后续不再更新）。
  - 因为2.0的接口和1.0完全不一样，不会导致冲突，可以同时使用，然后逐渐下线1.0，保证系统稳定。
- 不足：

  需要修改代码，虽然改动比较小（主要是初始化的类，以及继承的processor接口），但是如果Job特别多，改动也会比较大。

方案二：兼容

使用迁移工具把SchedulerX 1.0的Job全量导入到2.0，然后使用SchedulerX 2.0的定制兼容包（schedulerx-worker-1.0.6-compatible），不需要修改代码。

- 优势：

  不需要修改任何配置和代码。

- 不足：
  - 后续无法升级，定制兼容包只会发布 `1.0.6-compatible` 这一个版本。
  - 因为兼容了1.0的接口，工程需要移除1.0的JAR包，完全用2.0替代，不能同时使用1.0和2.0。
  - 无法使用2.0新功能。如果想使用新功能，建议使用重建方案迁移。

前提条件

1. 创建任务分组。
   i. 登录。
   ii. 在分布式任务调度 2.0 应用管理页面选择迁移的目的**地域**和**命名空间**。

   iii. 在**创建应用分组**对话框输入**应用名**和**Group ID**，然后单击**确定**。

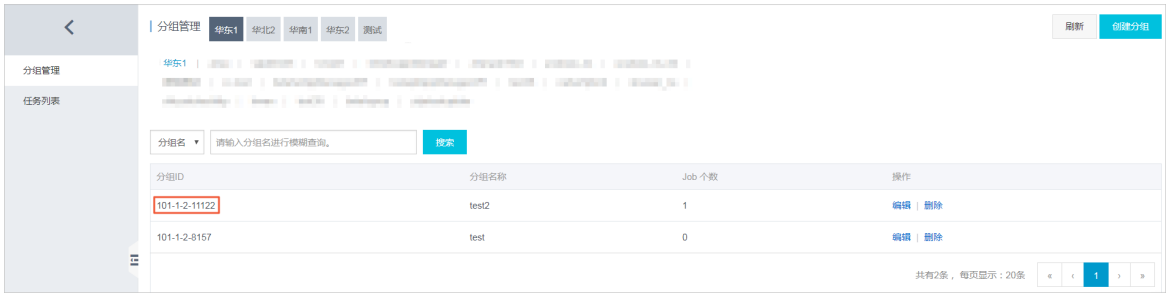
   Group ID会因两种迁移方案而不同。
   - 如果使用重建方案，可以根据业务自定义Group ID，如 `xxx.defaultGroup` 。
   - 如果使用兼容方案，Group ID需要和要迁移的1.0的Group ID保持一致，这样不用修改任何配置和代码。Group ID可以在Scheduler 1.0 分组管理页面查看。

   iv. 创建完成后，返回**应用管理**页面查看任务分组是否成功。
2. 下载迁移工具（JAR 包），通过命令行进入迁移工具所在目录，并执行 `java -jar schedulerx-migrate-1.0.20190729.jar` 命令。

阿里云1.0非测试环境迁移到阿里云2.0非测试环境

在迁移工具的命令行交互页面根据提示完成迁移。

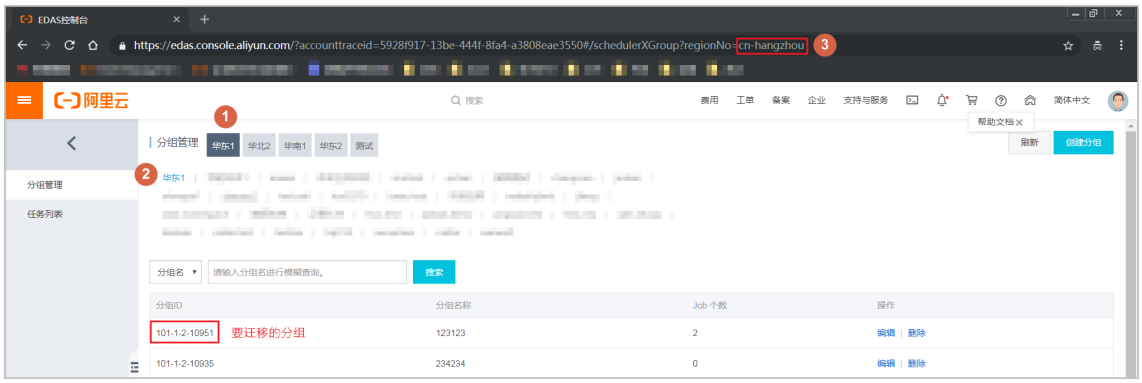
1. 输入迁移类型（请输入 *2*）。
2. 输入分布式任务系统1.0需要迁移分组ID。 分组ID可以在Scheduler 1.0 分组管理页面查看。



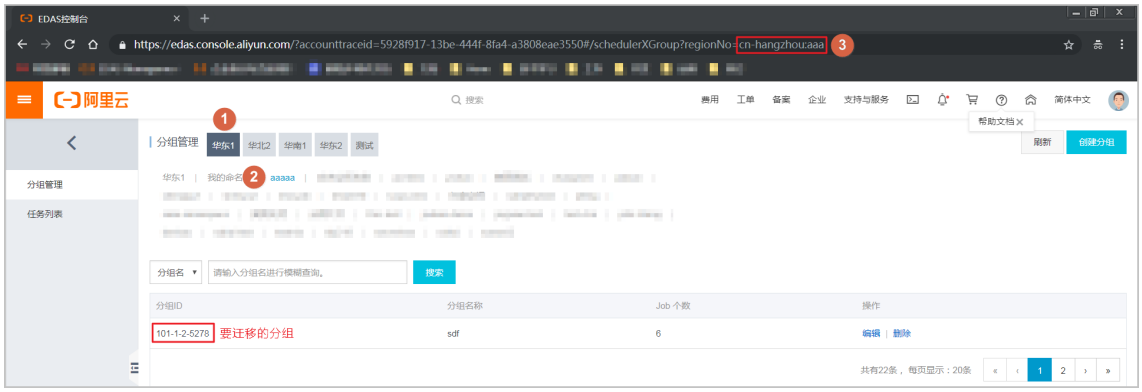
3. 输入分布式任务系统1.0迁移分组所在的 `regionId` （包含对应namespace的key）。

   在控制台进入分布式任务管理（SchedulerX 1.0）的**分组管理**页面，选择要迁移的分组所在的**地域**和**命名空间**，在URL中获取 `reginoNo` 的值（namespace的key即URL中的 `regionNo` ）。

   - 如果在默认命名空间中创建任务分组， `regionNo` 中只有Region，没有Namespace，如 `cn-hangzhou` 。



   - 如果在非默认命名空间中的任务分组， `regionNo` 会包含Region和Namespace，如 `cn-hangzhou:aaa` 。



4. 输入分布式任务系统1.0迁移分组所在的命名空间的 `tid` 。

   迁移分组所在的命名空间的 `tid` 。可以在EDAS 控制台的命名空间页面获取。

5. 输入分布式任务系统1.0迁移分组所属主账号的ID。

   需要填写能够看到该1.0任务分组的主账号ID。可以在账号管理控制台的安全设置页面中获取。

6. 输入迁入到分布式系统2.0分组的 `Group_ID` 。

   填写在使用创建应用设置的Group ID。

   > **说明** 如果是兼容迁移，2.0中的Group ID要和1.0中的Group ID一致。

7. 输入迁入到分布式系统2.0分组的主账号ak和SK。

   填写在SchedulerX 2.0中创建任务分组或有该分组操作权限的云账号的AccessKey ID和Access Key Secret。

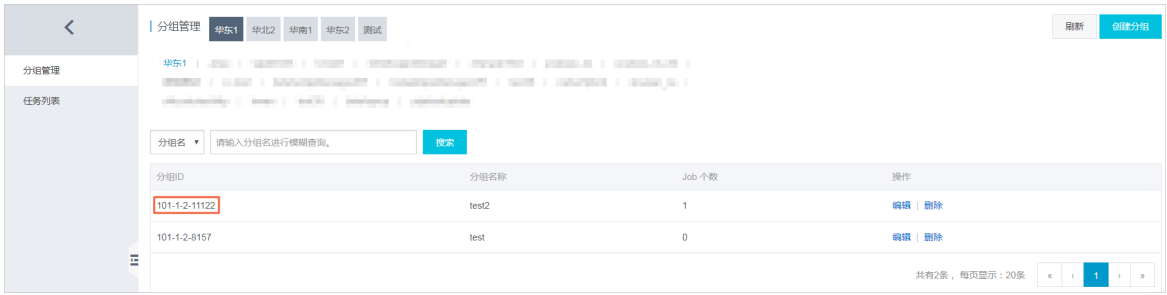
8. 输入迁入到分布式系统2.0分组对应的命名空间 `tid` 。 如果SchedulerX 2.0和1.0的命名空间的 `tid` 相同，可以不填，直接单击**回车**跳过。

9. 设置完成，单击**回车**，开始执行迁移。

阿里云1.0测试环境迁移到阿里云2.0测试环境

在迁移工具的命令行交互页面根据提示完成迁移。

1. 输入迁移类型（请输入 *3*）。
2. 输入分布式任务系统 1.0 需要迁移分组 ID。 分组 ID 可以在Scheduler 1.0 分组管理页面查看。



3. 输入分布式任务系统 1.0 迁移分组所属主账号的 ID。

   需要填写能够看到该 1.0 任务分组的主账号 ID。可以在账号管理控制台的安全设置页面中获取。

4. 输入要迁入到分布式系统2.0的 `regionName` （*测试（华东1）*）。
5. 输入迁入到分布式系统2.0分组的 `Group_ID` 。

   填写在使用创建应用设置的Group ID。

> **说明** 如果是兼容迁移，2.0中的Group ID要和1.0中的Group ID一致。

6. 请输入要迁入SchedulerX 2.0对应分组**appkey**。 输入上一步中应用管理列表里面 `Group_ID` 对应的**appkey**。

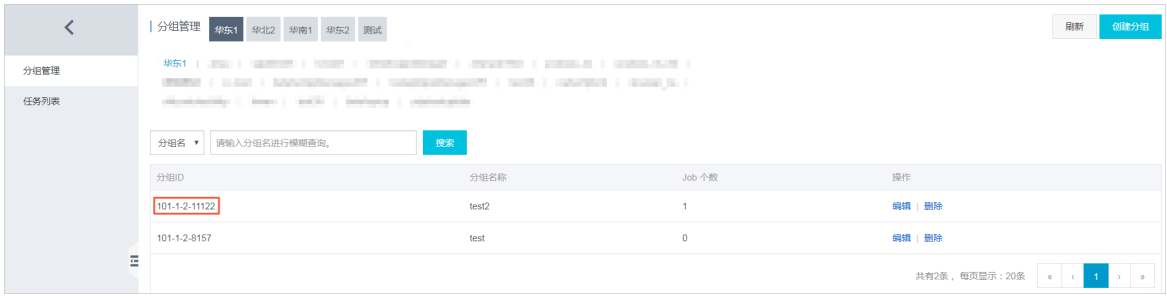
7. 输入迁入到分布式系统2.0分组对应的命名空间 `tid` 。 在命名空间页面查看对应的列表。输入测试的命名空间ID。

8. 设置完成，单击**回车**，开始执行迁移。

阿里云1.0测试环境迁移到阿里云2.0其它正式环境

在迁移工具的命令行交互页面根据提示完成迁移。

1. 输入迁移类型（请输入 *3*）。
2. 输入分布式任务系统 1.0 需要迁移分组 ID。 分组 ID 可以在Scheduler 1.0 分组管理页面查看。



3. 输入分布式任务系统 1.0 迁移分组所属主账号的 ID。

   需要填写能够看到该 1.0 任务分组的主账号 ID。可以在账号管理控制台的安全设置页面中获取。

4. 输入要迁入到分布式系统2.0的 `regionName` 。

   在SchedulerX 2.0选择您要迁入的地域名称。直接复制区域名称即可。

5. 输入迁入到分布式系统2.0分组的 `Group_ID` 。

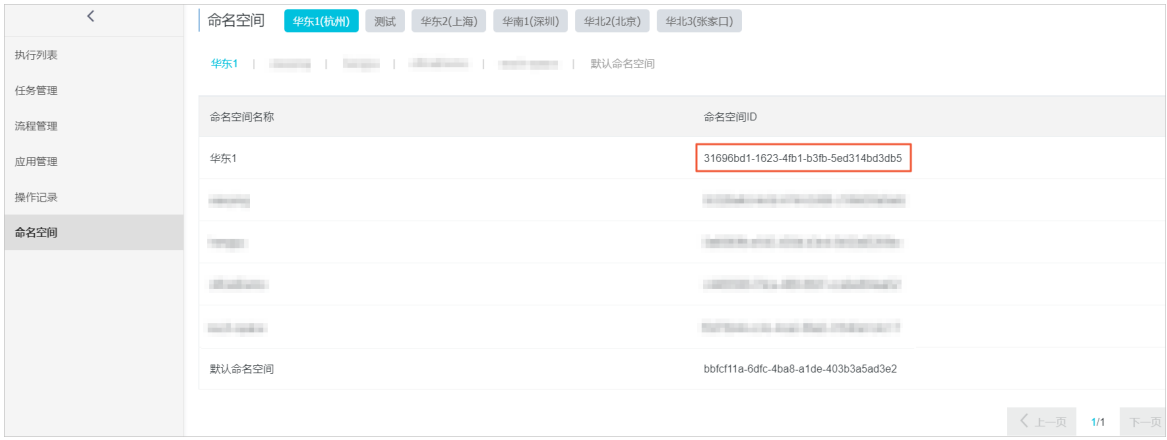
   填写在使用创建应用设置的Group ID。

   > **说明** 如果是兼容迁移，2.0中的Group ID要和1.0中的Group ID一致。

6. 输入迁入到分布式系统 2.0 分组的主账号 ak 和 SK。

   填写在 SchedulerX 2.0 中创建任务分组或有该分组操作权限的云账号的 AccessKey ID 和 Access Key Secret。

7. 输入迁入到分布式系统 2.0 分组对应的命名空间 `tid` 。 在命名空间页面查看对应的列表。输入测试的命名空间 ID。



8.

执行结果

- 在迁移工具中验证迁移结果

  执行结束后命令行页面会提示此次的迁移任务总数和失败、成功的任务数。

```
总共尝试迁移任务: 4 个
成功迁移: 4 个
Disconnected from the target VM, address: '127.0.0.1:49771', transport: 'socket'
```

  如果失败，还会提示失败的原因。并可以到到 `${user.home}/logs/schedulrx2-migrate/migrate.log` 日志中搜索具体jobid查看失败原因。

  > **说明** 同一个分组多次传输需要在同一台机器上操作，所有的成功信息都会存放到本地文件中，请勿删除，具体位置在 `${user.home}/migrate_jog` 中。

  如果不选择同一台机器，任务会重复迁移，文件被删除，第二次迁移还是会全量迁移。

- 升级JAR包
  i. 如果任务量比较多，把schedulerx-client的JAR包替换为：

```
<dependency>
<groupId>com.aliyun.schedulerx</groupId>
<artifactId>schedulerx2-worker</artifactId>
<version>1.0.6-compatible</version>
</dependency>
```

  ii. 如果任务量比较少，可以在SchedulerX 2.0控制台手动重建任务，并把schedulerx-client的JAR包替换为：

```
<dependency>
<groupId>com.aliyun.schedulerx</groupId>
<artifactId>schedulerx2-worker</artifactId>
<version>1.0.6</version>
</dependency>
```

- 登录SchedulerX 2.0的控制台，验证接入2.0成功，非常重要。
  - 在**应用管理**页面单击**连接机器**，能看到机器列表。
  - 在**任务管理**页面单击**运行一次**，能触发到业务代码。
  - 运行一次之后，在**执行列表**能看到触发记录。

2.常见问题

您在使用SchedulerX过程中会碰到一些问题。本文将介绍如何处理一些典型的使用问题。

机器繁忙

- 现象

  在应用管理页面某个分组的**操作**列单击**查看实例**，**连接实例**面板中该机器的**状态**为 `繁忙` 。

- 可能的原因
  - 机器的load5（负载）超过在创建任务分组时设置的阈值。
  - 机器的**内存使用率**超过在创建任务分组时设置的阈值。
  - 机器的**磁盘使用率**超过在创建任务分组时设置的阈值。
- 影响

  机器状态为繁忙，将导致无法触发调度任务。
- 处理方法

  i. 在**连接实例**面板中，将光标放到**繁忙**上，查看机器的load5、**内存使用率**和**磁盘使用率**。

  ii. 根据load5、**内存使用率**和**磁盘使用率**的值，确认繁忙的原因并进行相应的处理。例如清理磁盘、释放内存或负载。

  如果处理之后，机器仍然繁忙，可以升级机器的配置。

  如果只是用于测试，机器繁忙也想继续触发任务，可以在**应用管理**页面的**操作**列单击**编辑**，然后修改**编辑应用分组**面板中的高级配置，打开**是否触发繁忙机器**。

暂无可用机器

- 现象

  当应用接入任务调度，通过连接机器验证是否成功时，弹出 `暂无可用机器` 的提示。

- 可能的原因
  - 应用接入任务调度失败
  - JAR包冲突
  - SchedulerXWorker配置错误

- 处理方法

  i. 登录部署应用的ECS实例，查看是否存在日志目录*/home/${user.home}/logs/schedulerx/worker.log*。

> **说明**
> - 如果是root账号启动的进程，日志在*/root/logs/schedulerx/worker.log*。
> - 如果是admin账户启动的进程，日志在*/home/admin/logs/schedulerx/worker.log*。

- 如果不存在，说明ECS实例没有接入SchedulerX客户端，请重新接入，详情请参见Java应用接入SchedulerX、Spring应用接入SchedulerX或Spring Boot应用接入SchedulerX。
- 如果存在，进行下一步。

ii. 查看 `worker.log` 日志是否有异常日志。

- 如果日志包含 `groupId is not existed` ，说明是配置问题。在*worker.log*里搜索*Schedulerx WorkerConfig*，可以看到当前的配置，然后和控制台对比、确认。

> **注意** namespace可以从**命名空间**页面的**命名空间ID**获取。

- 如果有异常日志，确认是否是JAR包冲突。详情请参见JAR包冲突。
- 如果没有异常日志，搜索“started”，检查SchedulerXWorker的配置是否正确。

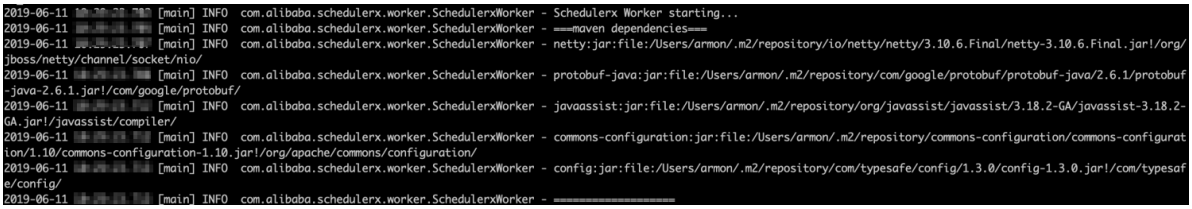
JAR包冲突

- 现象

  部署调度任务时，提示 `JAR包冲突` 。

- 可能的原因

  JAR包冲突可能包含其中具体子包冲突的多种可能的原因。 可以在日志里搜一下maven dependencies，可以看到每个JAR的版本和路径，帮助快速定位和解决JAR包冲突，如下图。



- 处理方法

  然后参照下表，升级低版本。

JAR包	版本
guava	20.0
com.typesafe.config	1.3.0
protobuf-java	2.6.1
io.netty	3.10.6.Final
javaassist	3.21.0-GA
hessian	4.0.51

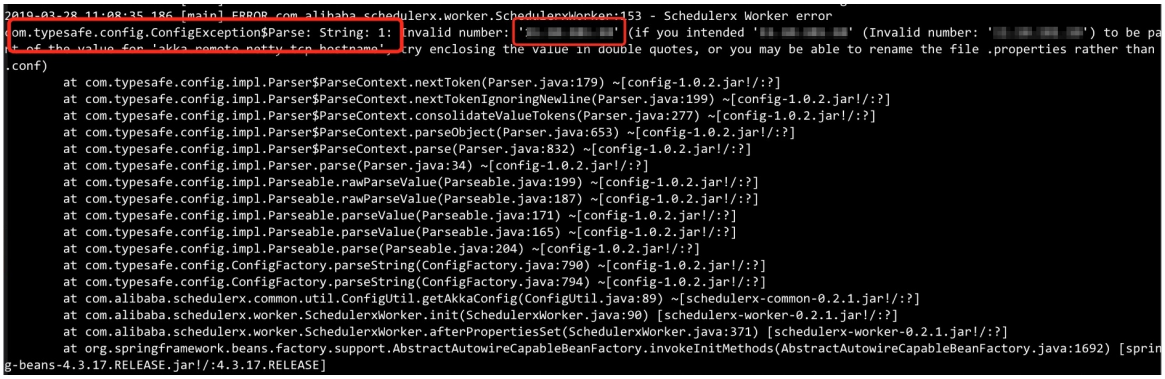
JAR包	版本
commons-configuration	1.10
commons-validator	1.4.0

- 示例

  下面列出几个典型的JAR冲突现象：

  - guava冲突

  - com.typesafe.config包冲突



- protobuf冲突

```
2019-03-20 14:49:41.742 [33471_69216-akka.actor.default-dispatcher-4] WARN  com.alibaba.schedulerx.worker.SchedulerxWorker:315 - heartbeat error
java.lang.VerifyError: class com.alibaba.schedulerx.protocol.Worker$WorkerHeartBeatRequest overrides final method getUnknownFields.()Lcom/google/protobuf/UnknownFieldSet;
at java.lang.ClassLoader.defineClass1(Native Method) ~[?:1.8.0_181]
at java.lang.ClassLoader.defineClass(ClassLoader.java:763) ~[?:1.8.0_181]
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142) ~[?:1.8.0_181]
```

```
    at scala.concurrent.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107)
2019-06-18 15:06:20.612 [24904_30202-akka.actor.default-dispatcher-194] WARN  com.alibaba.schedulerx.worker.SchedulerxWorker - active server=[redacted]:8080 lost.
java.util.concurrent.TimeoutException: Futures timed out after [5 seconds]
    at scala.concurrent.impl.Promise$DefaultPromise.ready(Promise.scala:223)
    at scala.concurrent.impl.Promise$DefaultPromise.result(Promise.scala:227)
    at scala.concurrent.Await$$anonfun$result$1.apply(package.scala:190)
    at akka.dispatch.MonitorableThreadFactory$AkkaForkJoinWorkerThread$$anon$3.block(ThreadPoolBuilder.scala:167)
    at scala.concurrent.forkjoin.ForkJoinPool.managedBlock(ForkJoinPool.java:3640)
    at akka.dispatch.MonitorableThreadFactory$AkkaForkJoinWorkerThread.blockOn(ThreadPoolBuilder.scala:165)
    at scala.concurrent.Await$.result(package.scala:190)
    at scala.concurrent.Await.result(package.scala)
    at com.alibaba.schedulerx.protocol.utils.FutureUtils.awaitResult(FutureUtils.java:39)
    at com.alibaba.schedulerx.worker.SchedulerxWorker$2.run(SchedulerxWorker.java:504)
    at akka.actor.LightArrayRevolverScheduler$$anon$2$$anon$1.run(LightArrayRevolverScheduler.scala:102)
    at akka.dispatch.TaskInvocation.run(AbstractDispatcher.scala:39)
    at akka.dispatch.ForkJoinExecutorConfigurator$AkkaForkJoinTask.exec(AbstractDispatcher.scala:415)
    at scala.concurrent.forkjoin.ForkJoinTask.doExec(ForkJoinTask.java:260)
    at scala.concurrent.forkjoin.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1339)
    at scala.concurrent.forkjoin.ForkJoinPool.runWorker(ForkJoinPool.java:1979)
    at scala.concurrent.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107)
2019-06-18 15:06:25.624 [24904_30202-akka.actor.default-dispatcher-194] WARN  com.alibaba.schedulerx.worker.SchedulerxWorker - active server=[redacted]:8080 lost.
java.util.concurrent.TimeoutException: Futures timed out after [5 seconds]
    at scala.concurrent.impl.Promise$DefaultPromise.ready(Promise.scala:223)
    at scala.concurrent.impl.Promise$DefaultPromise.result(Promise.scala:227)
    at scala.concurrent.Await$$anonfun$result$1.apply(package.scala:190)
```

- Netty冲突导致ActorSystem.create超时导致启动不起来

```
2019-03-20 11:52:48.030 [main] ERROR com.alibaba.schedulerx.worker.SchedulerxWorker:142 - Schedulerx Worker error
java.util.concurrent.TimeoutException: Futures timed out after [10000 milliseconds]
at scala.concurrent.impl.Promise$DefaultPromise.ready(Promise.scala:223) ~[scala-library-2.11.11.jar:?]
at scala.concurrent.impl.Promise$DefaultPromise.result(Promise.scala:227) ~[scala-library-2.11.11.jar:?]
at scala.concurrent.Await$$anonfun$result$1.apply(package.scala:190) ~[scala-library-2.11.11.jar:?]
at scala.concurrent.BlockContext$DefaultBlockContext$.blockOn(BlockContext.scala:53) ~[scala-library-2.11.11.jar:?]
at scala.concurrent.Await$.result(package.scala:190) ~[scala-library-2.11.11.jar:?]
at akka.remote.Remoting.start(Remoting.scala:189) ~[akka-remote_2.11-2.4.20.jar:?]
at akka.remote.RemoteActorRefProvider.init(RemoteActorRefProvider.scala:212) ~[akka-remote_2.11-2.4.20.jar:?]
at akka.actor.ActorSystemImpl.liftedTree2$1(ActorSystem.scala:828) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystemImpl._start$lzycompute(ActorSystem.scala:825) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystemImpl._start(ActorSystem.scala:825) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystemImpl.start(ActorSystem.scala:841) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystem$.apply(ActorSystem.scala:245) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystem$.apply(ActorSystem.scala:288) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystem$.apply(ActorSystem.scala:263) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystem$.create(ActorSystem.scala:191) ~[akka-actor_2.11-2.4.20.jar:?]
at akka.actor.ActorSystem.create(ActorSystem.scala) ~[akka-actor_2.11-2.4.20.jar:?]
at com.alibaba.schedulerx.worker.SchedulerxWorker.init(SchedulerxWorker.java:119) [schedulerx-worker-0.2.0.jar:?]
```

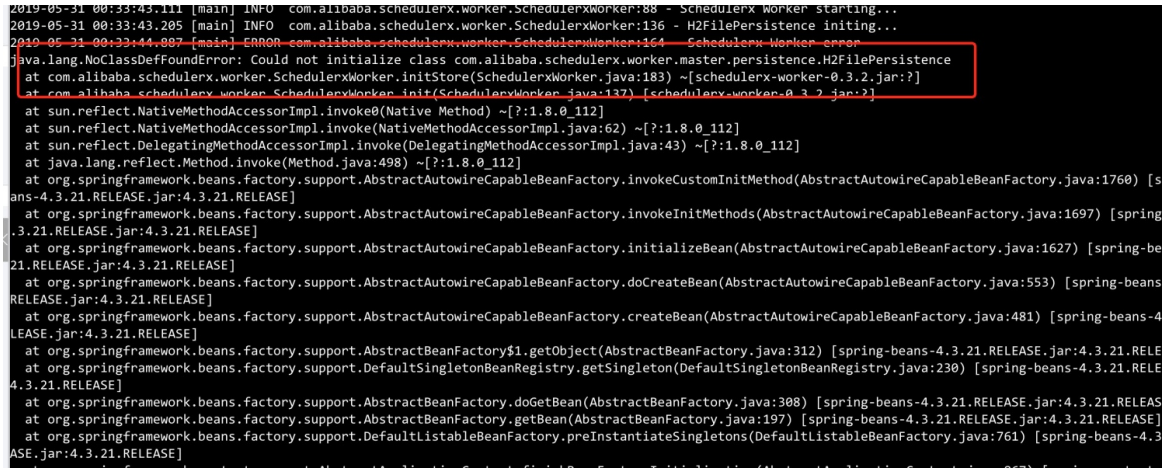
- Hessian冲突

```
java.lang.NoSuchMethodError: com.caucho.hessian.io.SerializerFactory.createDefault()Lcom/caucho/hessian/io/SerializerFactory;
```

- commons-validator冲突

```
Caused by: java.lang.NoClassDefFoundError: org/apache/commons/validator/routines/InetAddressValidator
  at com.aliyun.openservices.log.util.NetworkUtils.getLocalMachineIP(NetworkUtils.java:33) ~[aliyun-log-0.6.31.jar:?]
  at com.aliyun.openservices.log.Client.<init>(Client.java:250) ~[aliyun-log-0.6.31.jar:?]
  at com.aliyun.openservices.aliyun.log.producer.ProjectConfigs.buildClient(ProjectConfigs.java:34) ~[aliyun-log-producer-0.2.0.jar:?]
  at com.aliyun.openservices.aliyun.log.producer.ProjectConfigs.put(ProjectConfigs.java:13) ~[aliyun-log-producer-0.2.0.jar:?]
  at com.alibaba.schedulerx.worker.logcollector.SlsLogCollector.<init>(SlsLogCollector.java:64) ~[schedulerx-worker-1.0.3.jar:?]
  at sun.reflect.GeneratedConstructorAccessor156.newInstance(Unknown Source) ~[?:?]
  at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) ~[?:1.8.0_161]
  at java.lang.reflect.Constructor.newInstance(Constructor.java:423) ~[?:1.8.0_161]
  at com.alibaba.schedulerx.common.util.ReflectionUtil.newInstance(ReflectionUtil.java:31) ~[schedulerx-common-1.0.2.jar:?]
```

- javaassist冲突



- httpclient冲突

tablestore protobuf-2.4.1和SchedulerX不兼容

详情请参见使用Java SDK时出现PB库冲突。

Spring应用找不到Bean

- 现象

  Spring应用找不到对应的Bean。

- 可能的原因
  - 接入方式不正确。
  - JobProcessor中未注入Bean。
  - *pom.xml*中添加了 `spring-boot-devtools` 依赖。
  - 在JobProcessor和process方法中添加了事务注解。
  - bean被其他切面代理。
  - classLoader不一致。
- 处理方法
  - 在应用管理页面该分组的**操作**列单击**连接机器**，**连接机器**面板中查看启动方式是否为Spring或Spring Boot。

    如果启动方式不是Spring或Spring Boot，请检查您的应用。
  - 检查应用代码中是否将JobProcessor注入为Bean，例如添加了@Component注解。

    如果JobProcessor没有注入为Bean，请将JobProcessor注入为Bean。
  - 检查*pom.xml*中是否添加了 `spring-boot-devtools` 。

    如果*pom.xml*中添加了 `spring-boot-devtools` ，请删除 `spring-boot-devtools` 依赖。
  - 检查JobProcessor和process方法中是否添加了事务注解。

    在JobProcessor和process方法中添加了事务注解，请删除注解。
  - 检查bean列表是否被其他切面代理。

    把断点打到*DefaultListableBeanFactory*类，检查*beanDefinitionNames*成员列表（即是spring注册的bean列表）是否被其他切面代理。

    如果bean列表被其他切面代理，请删除代理。
  - 调试*ThreadContainer.start*，查看是否报class.forName错。

    如果是报class.forName错，但是class又真实存在，可以确定是classLoader的问题，可能是您使用了某框架导致classLoader不一致。请通过SchedulerxWorker.setClassLoader解决。

  如果以上方法都未能解决问题，请联系SchedulerX技术支持人员。

tablestore protobuf-2.4.1和SchedulerX不兼容

请参见使用 Java SDK 时遇到 Protobuf 或 HttpClient 库冲突。

submit jobInstanceId to worker timeout

- 现象

```
submit jobInstanceId=28384771 to worker=WorkerInfo [ip=192.168.32.166,
port=41114, workerId=30574_7365, metrics=Metrics(cpuLoad1=0.01, cpuLoad5=0.05600000000000001,
cpuProcessors=2, heap1Usage=0.20147679324894516, heap5Usage=0.20379746835443036,
heap1Used=191, heapMax=948, diskUsage=0.24241670191853087, diskUsed=9742, diskMax=40187)] timeo
ut, cost=5004ms
```

- 可能的原因
  - JAR包冲突
  - 机器负载过高或者重启

- 解决办法

  查看*worker.log*中有没有异常日志。

  - 如果有，可能是JAR包冲突，处理详情请参见JAR包冲突。
  - 如果没有，可能是机器负载过高或者重启造成的，请检查机器。

重启应用后，任务卡住

- 现象

  应用发布没问题，通过Aone重启应用，任务会卡住。

- 可能的原因

  如果启动进程的系统环境变量为 `user.dir="/"` ，但是启动进程的账号不是root，会造成SchedulerX底层消息通知失败。原因是没有权限写*akka.persistence.snapshot*目录。

- 解决办法

  将SchedulerX升级到1.0.9-hotfix-v3及以上版本。

任务运行中卡住

- 现象

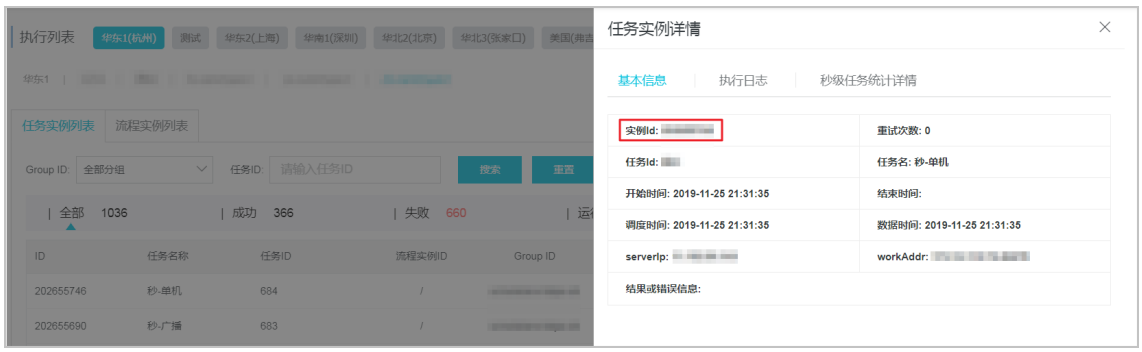
  调度任务一直处于执行中，不能结束。

- 可能的原因
  - 业务的问题
  - SchedulerX的问题

- 解决方法

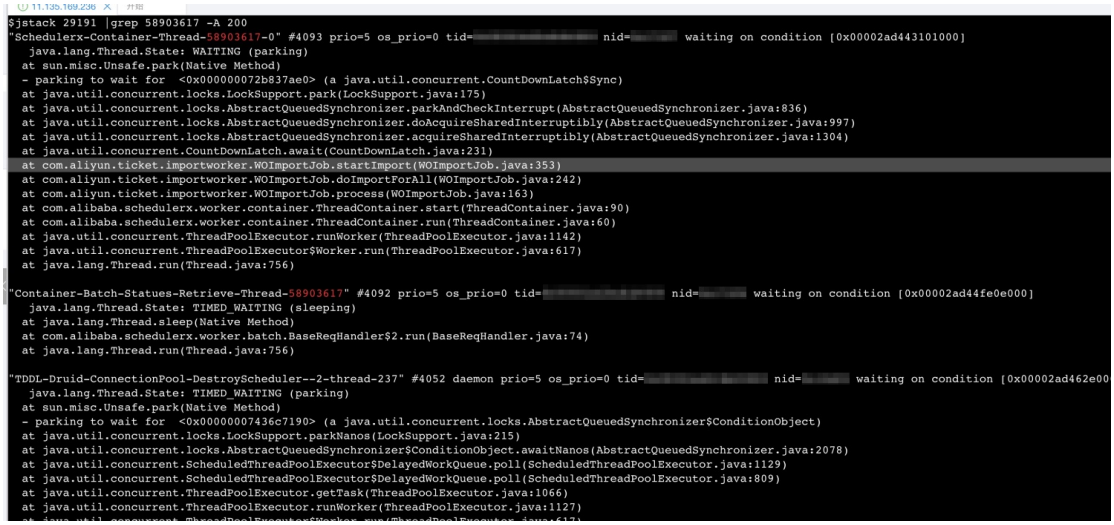
  `schedulerx-worker` 执行每个processor的时候会把任务实例ID放到线程名中，方便查看线程栈。这里以分布式任务某个子任务卡住为例，单机执行/广播执行类似的解决方法类似。

  i. 在控制台执行列表页面查看卡住的任务实例的详情，获取实例ID。

  ii. 登录卡住的机器，执行**jstack [pid] | grep [实例id] -A 20**，查看线程栈。

- 如果执行结果和下图相似，说明是业务的问题。



- 否则，请联系SchedulerX技术支持人员。

报slf4j.Slf4jMDC异常

- 现象

- 解决方法

  排除掉schedulerx2.0依赖的slf4j。

org.apache.commons.logging.impl.LogFactoryImpl类找不到问题

背景

Schedulerx2.0启动时会按一定策略查找 `LogFactory` 的实现类，其中的一个策略是扫描配置文件*META-INF/services/org.apache.commons.logging.LogFactory*，该配置文件的第一行会定义 `LogFactory` 的实现类名。

现象示例

`com.taobao.common.division:common-division:3.0.28` 依赖的 `com.alibaba.common.logging:toolkit-common-logging:1.0` 这个JAR包下有一个配置文件*META-INF/services/org.apache.commons.logging.LogFactory*，该配置文件中声明了LoggerFactory的实现类 `com.alibaba.common.logging.spi.GenericLoggerFactory`；该实现类依赖了 `org.apache.commons.logging.impl.LogFactoryImpl` 这个具体实现类，而具体实现类是由 `commons-logging` 这个三方包提供的，如果应用排除掉了这个三方包，那么就会遇到 `java.lang.ClassNotFoundException` 的问题。

RAM用户无权限进行应用管理

- 现象

以RAM用户登录控制台，单击**应用管理**，提示**没有权限**。

- 解决办法

  使用阿里云账号为该RAM用户添加自定义权限，权限内容如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "ram:ListUsers",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

3.报警常见问题

在使用SchedulerX时，您可能会收到一些报警。本文介绍如何处理这些报警。

killed from server

- 报警信息

  您[redacted]环境的定时调度任务 namespace: 默认空间,jobid:[redacted], 触发实例id:[redacted], 在2020-03-17 07:40:57本次触发失败, 运行机器[redacted]:37761, 原因可能是killed from server, 请及时关注。[SchedulerX2访问 [redacted].com]
  alarmTime=2020-03-17 10:40:57

- 报警说明

  这个报警通常是因为配置了超时自动kill。

don't update progress more than 30s

- 报警信息

  您[redacted]环境的定时调度任务 namespace: 默认空间,jobid:[redacted]是否完成判断_BACKUP1), 触发实例id:[redacted], 在2020-03-17 10:59:00本次触发失败, 运行机器[redacted]:42055, 原因可能是jobInstance=[redacted] don't update progress more than 30s, maybe worker=[redacted]:42055 is down., 请及时关注。[SchedulerX2访问 [redacted].com]
  alarmTime=2020-03-17 11:01:13

- 报警说明

  客户端和服务端失联（超过30秒没有发送心跳），一般来说是因为客户端发布或者负载太高导致的。如果是发布或者重启导致的，可以配置任务失败自动重试。