ALIBABA CLOUD

阿里云

分布式任务调度 SchedulerX

产品简介

文档版本：20200909

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

1.什么是分布式任务调度SchedulerX

分布式任务调度SchedulerX是阿里巴巴基于Akka架构自研的新一代分布式任务调度平台，提供定时调度、调度任务编排和分布式批量处理等功能。

您可以在控制台配置、管理您的定时调度任务、查询任务执行记录和运行日志，还可以通过工作流进行任务编排和数据传递。SchedulerX提供了简单、易用的分布式编程模型，通过简单几行代码就可以将海量数据分发到多台机器上执行。

如果您有关于分布式任务调度SchedulerX的任何疑问，欢迎加入钉钉答疑群咨询：23103656。

2.产品功能

SchedulerX主要提供调度、执行和运维三方面的功能。

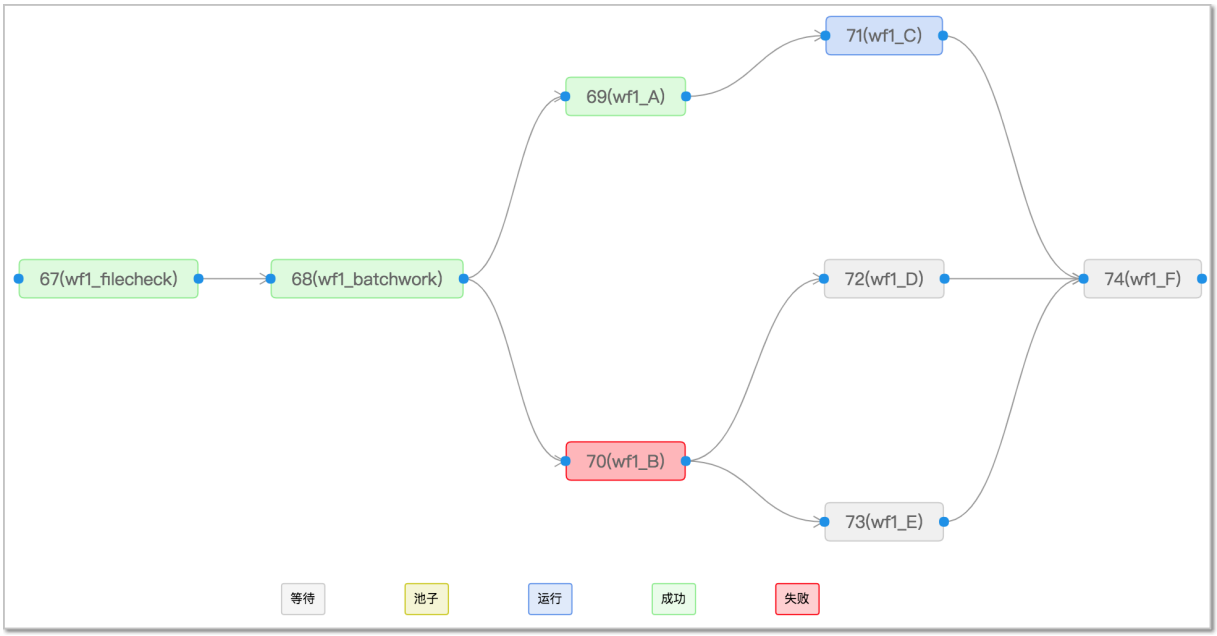
多种表达式的定时调度

- Crontab：支持Unix Crontab表达式，详情请参见Cron。不支持秒级别。
- Fixed rate：Crontab必须被60整除，不支持其它数量级时间间隔的任务，如每隔40分钟的定时任务。Fixed rate专门用来做定期轮询，可以弥补Crontab的不足，且表达式简单，详情请参见Fixed rate。不支持秒级别。
- Second delay：适用于对实时性要求比较高的业务，例如执行间隔为10秒的定时调度任务，详情请参见Second delay。支持秒级别。
- 日历：支持多种日历，还可以自定义导入日历。适用于金融业务，如需要在每个交易日执行定时任务。
- 时区：适用于跨国业务，如需要在每个国家所在时区执行定时任务。

创建定时调度任务的操作步骤请参见创建调度任务。

调度任务编排

使用有向无环图DAG（Directed Acyclic Graph）进行任务编排，操作简单，前端直接拖拖拽拽即可。详细的任务状态图能直观的看到下游任务为什么没执行。详情请参见创建工作流。



多种调度任务类型

在定时调度和工作流调度中支持基于多语言的多种任务类型。

- Java：可以在用户进程中执行，也可以通过上传JAR包动态加载。详情请参见Java任务。
- Shell：前端直接写Shell脚本。详情请参见脚本任务。
- Python：前端直接写Python脚本，需要Python环境。详情请参见脚本任务。
- Go：前端直接写Go脚本，需要Go环境。详情请参见脚本任务。
- HTTP（Serverless）：支持Serverless的HTTP任务，包含GET和POST两种方法，无需依赖Client，在控制台配置完即可生效使用。详情请参见HTTP任务（Serverless）。

- 自定义：可以自定义任务类型，然后实现一个Plugin即可。

分布式计算

提供简单、易用的分布式编程模型，可以进行大数据跑批。

- 单机：随机挑选一台机器执行。详情请参见单机。
- 广播：所有机器同时执行且等待全部结束。详情请参见广播。
- Map模型：类似于Hadoop MapReduce里的Map。只要实现一个Map方法，简单几行代码就可以将海量数据分布式到多台机器上执行。详情请参见Map模型。
- MapReduce模型：MapReduce模型是Map模型的扩展，废弃了postProcess方法，新增Reduce接口。所有子任务完成后会执行Reduce方法，可以在Reduce方法中返回该任务实例的执行结果，或者回调业务。详情请参见MapReduce模型。
- 分片运行：类似Elastic-Job模型，控制台配置分片参数，可以将分片平均分给多个客户端执行。支持多语言版本。详情请参见多语言版本分片模型。

资源管理和任务优先级

通过应用级别资源管理，可以控制一个应用同时运行的最大任务数量。再通过任务优先级，可以实现类似于yarn的任务优先级队列，超过并发数的任务在队列中等待，高优先级任务会抢占低优先级任务优先调度。

很多场景下都有应用级别资源控制和任务优先级的需求。例如数据平台每天要收集报表，可能会有成千上万的任务在夜间执行。如果没有资源控制，所有任务一起执行会导致应用不可用。运营报表又必须在早上9点前生成，这就需要在资源控制的基础上，高优先级任务优先调度。如果低优先级任务先进入队列，高优先任务也能抢占优先调度。

运维能力

- 数据大盘：控制台提供了执行记录大盘和执行列表，可以看到每个任务的执行历史，并提供操作。
- 查看日志：每次执行的调度任务都可以在详情中查看运行日志。如果任务执行失败，前端直接就能看到错误日志，非常方便。详情请参见查看任务实例详情。
- 原地重跑：任务失败，修改完代码发布后，可以立即重新执行。
- 标记成功：任务失败，如果后台把数据处理修正了，重新执行又需要几个小时，可以直接将任务标记为成功。
- 停止调度任务：实现JobProcessor的 `kill()` 接口，您就可以在前端停止正在运行的任务，甚至子任务。

数据偏移时间

SchedulerX可以处理有数据状态的任务，在创建任务的时候设置调度时间，而实际上处理的数据时间可能和任务执行时间不一致，可以配置时间偏移，调度时间 + 时间偏移即数据时间。例如一个任务是每天00:30运行，但是实际上要处理前一天的数据，就可以向前偏移一个小时。调度时间不变，执行的时候通过 `context.getDataTime()` 获得的就是前一天23:30。

重刷数据

既然任务具有了数据时间，就会用到重刷数据。例如一个工作流最终产生一个报表，但是业务发生变更（新增一个字段）或者发现上一个月的数据有错误，那么就需要重刷过去一个月的数据。通过重刷数据功能，可以重刷某些任务/工作流的数据（只支持天级别），每个实例都是不同的数据时间。详情请参见重刷调度任务数据。

失败自动重试

- 实例失败自动重试：在任务管理的高级配置中，可以配置实例失败重试次数和重试间隔，例如重试3次，

每次间隔30秒。如果重试3次仍旧失败，该实例状态才会变为失败，并发送报警。

- 子任务失败自动重试：如果是分布式任务（并行计算/内网网格/网格计算），子任务也支持失败自动重试和重试间隔，同样可以通过任务管理的高级配置进行配置。

报警监控

- 失败报警
- 超时报警
- 无可用机器报警
- 报警方式：短信

3.产品优势

SchedulerX有高可靠、高性能、节约成本、提升效率和安全防护等优势。

高可靠

通过分布式架构、数据三备份、消息At-least-once delivery、Failover和定期轮检等手段，保证任务调度和运行的高可靠。

高性能

支持秒级别调度，轻量级分布式计算可以帮助您完成准实时的大数据跑批。

节约成本和提升效率

无机器和人工运维成本，接入简单，提供报警监控。

安全防护

多层次安全防护，包括：

- 支持HTTPS，VPC访问。
- 支持用户隔离、命名空间隔离和应用隔离。
- 支持RAM子账号和临时AK。

4.名词解释

本文主要对SchedulerX涉及的专有名词及术语进行定义和解释，方便您更好地理解相关概念并使用SchedulerX。

AppGroup
即应用分组，映射用户的具体应用，关联绑定机器，用来做业务的隔离。

DAG
Directed Acyclic Graph，即有向无环图。所谓有向无环图是指任意一条边有方向，且不存在环路的图。

Job
即任务，Job是SchedulerX中调度的最小单位。

Job instance
即任务实例，Job每次调度会产生一个JobInstance。

Namespace
即命名空间，SchedulerX提供的资源隔离服务，不同命名空间之间逻辑上天然隔离。命名空间帮助您将多个环境间的资源完全隔离，并可以使用一个账号进行统一管理。

Task
即子任务，并行计算/内存网格/网格计算，通过Map方法会产生Task。

Work Flow
即工作流，Work Flow是一个DAG（有向无环图），用来做任务编排。

调度时间
JobInstance每次调度的时间叫做调度时间，JobProcessor可以根据`context.getScheduleTime()`获取。

数据时间
SchedulerX可以处理有数据状态的任务。创建任务的时候可以填数据偏移。例如一个任务是每天00:30运行，但是实际上要处理上一天的数据，就可以向前偏移一个小时。运行时间不变，执行的时候通过`context.getDataTime()`获得的就是23:30（前一天）。

5.版本说明

5.1. 服务端版本说明

本章节介绍了SchedulerX服务端的发布记录及相关的功能。

2020-08-19

- 新特性
  - 支持一个应用10万+任务
  - 任务管理新增查看详情功能
- 优化
  - 搜索优化
    - 应用列表下拉框小于10个默认为全部应用，超过10个默认选中第一个应用。
    - 服务端部分查询、加载接口性能优化
  - 无可用机器报警可区分是否指定了机器
- 问题修复
  - 修复禁用任务后，池子状态的实例不会变为终止状态的问题
  - 修复数据库异常导致池子中的实例会一直卡住的问题
  - 修复删除应用后，再重新创建同名的应用会失败的问题
  - 修复用户重启后，秒级任务会停止调度的问题

2020-05-27

- 新特性
  - HTTP任务增强
    - HTTP任务支持Post参数
    - HTTP任务支持通过header获取任务基本信息
    - HTTP任务超时时间上限支持到30秒
  - POP API增强，新版本 `aliyun-java-sdk-schedulerx2-1.0.3`
    - 支持通过POP API创建HTTP任务
    - 支持授权/取消权限的POP API
  - 支持连续失败次数报警
  - 支持命名空间下的应用授权
- 优化
  - 搜索优化
    - 任务管理和流程管理支持通过状态搜索过滤
    - 执行列表支持通过实例ID搜索
    - 应用列表下拉框支持模糊搜索
  - 删除非空应用做检验，有任务不准删除应用

- HTTP任务性能优化
- 失败报警频率优化，增加疲劳度
- 控制台增加“联系我们”
- 应用名增加非中文检验

- 问题修复
  - 修复数据偏移无法设置负数
  - 修复超时时间等没有单位的前端bug
  - 修复服务端切换到主服务器时，指定机器会丢失的bug

5.2. 客户端版本说明

本章节介绍了SchedulerX客户端的发布记录及相关的功能。

1.2.0.1, 2020-08-19

- 新特性
  - 支持一个应用10万+任务（仅公共云支持）。
  - 新增客户端日志开关，默认开启。
  - OpenAPI创建任务，支持设置状态。
  - 去除diamond-client、logger.api和log4j依赖。
- 问题修复
  - 修复客户端断网演练会和服务端失联的bug。
  - 修复在EDAS中部署的应用读不到AccessKey的bug。

1.1.4.RELEASE, 2020-05-15

- 新特性
  - 支持自建Namespace。
  - 支持初始化多个schedulerxWorker。
  - MapReduce模型增强
    - 子任务失败，也能执行reduce。
    - JobContext.getTaskStatuses可以判断每个Task的状态，Map<Long, TaskStatus>结构体key是taskId，value是task的状态。
- 问题修复
  - ProcessResult，result为空，会导致空指针。
  - thread-dispatcher-delivery挂了会导致任务卡住。

1.1.2.RELEASE, 2020-02-10

- 新特性

  shade protobuf and netty from akka，解决接入时90%以上的JAR包冲突。
- 问题修复

  appKeys不支持多分组。

1.1.0, 2019-12-17

- 新特性
  - 支持多语言版本的分片模型（类似于elastic-job）。
  - OpenApi创建分组，可以返回appKey。
  - 成功状态支持重跑，工作流中的任务实例重跑任务本身及下游任务。
- 优化
  - Server端性能优化，和客户端通信方式由同步改为异步，并优化了AKKA默认Dispatcher的配置。
  - 使用1.1.0版本客户端，心跳性能优化提高3倍。
  - 前端任务管理列表重新设计，可以看到更多信息。
- 问题修复
  - 分布式拉模型，全局子任务可能不起作用。
  - 隔离单元环境中“如果没有配置domain，可能还是会启动失败。

1.0.9, 2019-11-28

- 新特性
  - 增加blockAppStart配置。表示SchedulerX启动失败是否block应用程序启动，默认true。
  - 新增查询工作流运行状态接口。接口为 `GetWorkflowInstanceRequest` 。
  - `jobContext` 上下文新增 `jobName` 字段。这样用户可以运行期间获取到任务名称。
- 问题修复
  - 通过Hessian反序列化BigDecimal为0；通过Hessian反序列化LocalDateTime报错。
  - 修复指定机器功能问题。任务运行超过一定时间子任务会下发到未指定机器上。
  - 客户端springContext.getBean报AnnotationConfigApplicationContext has not been refreshed yet异常。
  - 修复任务实现类配置错误的情况下会触发Spring Boot的 ServeletWebServer停止的逻辑，导致业务进程在，但是Web服务被shutdown问题。
  - 修复系统启动变量 `user.dir='/'` ，任务100%会卡住的问题。
  - 客户端springContext.getBean报AnnotationConfigApplicationContext has been closed already异常。
  - 客户端生成的 `workerId` 存在小概率重复冲突的问题，造成任务触发到非本应用的机器上。
  - Spring应用不能自定义class loader。
  - 秒级别任务广播执行计数器显示不对。
  - 秒级别任务，jobContext.getScheduleTime没有跟着循环更新。

1.0.8, 2019-08-06

- 新特性
  - 【重要】重构 `JobProcessor.postProcess` 接口，增加 `ProcessResult` 返回值，之前用到 `postProcess` 接口需要改代码。
  - 广播执行增强， `BroadcastJobProcessor` 支持 `preProcess` 和 `postProcess` 。 `preProcess` 会在所有机器执行 `process` 之前执行一次， `postProcess` 会在所有机器执行process后执行一次。

- JobContext.getTaskAttempt 可以获取当前子任务重试次数。
- 客户端支持自定义监听端口，例如 SchedulerxWorker.setPort 。
- Java任务可以实现 JobProcessor ，不必须继承 JavaProcessor 。

- 问题修复
  - 修复 taskId=1 的子任务不支持子任务自动重试的bug。
  - 分布式任务，根任务失败，无法看到失败原因。
  - 并行任务子任务列表不能重试子任务。

1.0.6-compatible, 2019-07-02

兼容 schedulerx1.0（DTS）接口的兼容版本。不支持同时依赖 schedulerx-client 和 schedulerx-worker 两个包，只能依赖 schedulerx-worker 一个包，即需要把DTS所有任务迁移到SchedulerX 2.0。

1.0.6, 2019-07-02

- 新特性
  - 新增部分包的 shade: aliyun-log 、 commons-validator 、 gson 、 fastjson 、 guava 和 commons-collections 。
  - 通过 ProcessResult(false, errorMsg) 返回，前端日志也能看到 errorMsg 。

- 优化
  - 优化at-least-once-delivery性能。
  - 高负载场景下，消息重复发送会造成秒级任务卡主或应用线程被Interrupt。

- 问题修复
  - 广播任务卡主问题。
  - 秒级任务卡主问题。
  - at-least-once-delivery可能会导致子任务状态无限重试。
  - logcollector初始化失败，异常抛出来，启动失败。
  - DB清理工作流任务实例，导致工作流无法恢复调度问题。
  - Spring方式启动，不支持kill。

1.0.3, 2019-06-06

- 新特性
  - MapReduce 模型支持返回所有子任务的结果，由Reduce处理。
  - 分布式模型支持拉模型，解决因为单机性能引起的木桶效应，支持动态扩容拉子任务。
  - 拉模型支持全局子任务并发度，可以进行限流。
  - JobContext增加wfInstanceId。
  - 客户端启动失败抛异常，堵塞JVM启动，尽早发现问题。
  - 客户端启动打印mvn依赖JAR的版本和路径，帮助排查JAR包冲突。
  - 分布式模型子任务详情，增加队列维度，可以看到每台机器缓存的子任务队列。

- 优化

- Worker因为Server负载高被误guarantined，可以自动恢复，不同重启客户端。
- 分布式模型子任务详情，运行中可以真实显示每台机器正在运行的子任务数。
- Master节点挂了，Server会负责清理Slave节点的资源，防止内存泄漏。

1.0.0, 2019-04-30

新特性

- 支持crontab和fixed_rate表达式进行周期性定时调度。
- 支持工作流调度，进行流程编排。
- 支持second_delay表达式进行秒级别调度。
- 支持Java、Shell、Python、Go任务类型。
- 支持单机执行、广播执行、并行计算、内存网格、网格计算。
- 支持Map和MapReduce分布式编程模型。
- 支持任务实例级别和子任务级别的失败自动重试（默认不重试）。
- 支持数据时间和重刷数据。

6.和开源产品对比

有开源产品同样可以实现分布式任务调度，本文介绍SchedulerX和开源产品的对比，帮助您更好的了解分布式任务调度和SchedulerX。

产品	Quartz	Elastic-Job	XXL-JOB	Apache Airflow	SchedulerX
定时调度	Cron	Cron	Cron	Cron	• Cron • Fixed rate • Second delay • OpenAPI
工作流	无	无	无	有，通过XML配置。	有，图形化配置，任务间可数据传递。
分布式任务	无	静态分片	静态分片	无	静态分片，MapReduce动态分片。
白屏化任务治理	无	• 执行记录：无 • 运行大盘：有 • 运行日志：有 • 原地重跑：无 • 重刷数据：无	• 执行记录：有 • 运行大盘：有 • 运行日志：有 • 原地重跑：无 • 重刷数据：无	• 执行记录：有 • 运行大盘：有 • 运行日志：有 • 原地重跑：有 • 重刷数据：有	• 执行记录：有 • 运行大盘：开发中 • 运行日志：有 • 原地重跑：有 • 重刷数据：有
任务类型	Java	• Java • Shell	• Java • Shell • Python • PHP • Node.js	可通过Operator自定义，自带的主要是大数据和Shell，无Java。	• Java • Shell • Python • Go • HTTP • Node.js • 自定义
报警监控	无	自研	邮件	自研	短信

产品	Quartz	Elastic-Job	XXL-JOB	Apache Airflow	SchedulerX
使用成本	• DB • 多个Server • 人工运维成本	• DB • 至少3个ZooKeeper • 2个Console • 多个Server • 人工运维成本	• DB • 1个调度中心 • 多个执行器 • 人工运维成本	• DB • 2个Master node • 多个Worker node • MQ • 人工运维成本	按照调度量和计算量收费，无机器和人工运维成本。