ALIBABA CLOUD

阿里云

企业级分布式应用服务 EDAS

企业级分布式应用服务EDAS公共云合集

文档版本：20220707

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.金融云用户指南概述 10
2.选择注册中心运维方式 15
3.资源检索 17
4.应用模板管理 21
5.多集群应用管理 24
6.分布式任务调度 27
6.1. 动态与公告 27
6.1.1. 功能发布记录 27
6.1.1.1. 2022年 27
6.1.1.2. 历史记录 29
6.1.2. 产品公告 36
6.1.2.1. 专业版商业化公告（2022年05月24日） 36
6.1.2.2. 专业版公测（2022年01月26日） 36
6.1.2.3. 基础版商业化公告（2021年07月30日） 38
6.2. 产品简介 38
6.2.1. 什么是分布式任务调度SchedulerX 38
6.2.2. 产品功能 38
6.2.3. 产品优势 40
6.2.4. 名词解释 41
6.2.5. 和开源产品对比 41
6.3. 产品计费 42
6.3.1. 计费说明 42
6.3.2. 欠费说明 44
6.4. 快速入门 45
6.4.1. 准备工作 45
6.4.1.1. 开通SchedulerX（免费） 45

6.4.1.2. 创建资源 45
6.4.2. 客户端快速接入SchedulerX 47
6.4.2.1. Java应用接入SchedulerX 47
6.4.2.2. Spring应用接入SchedulerX 49
6.4.2.3. Spring Boot应用接入SchedulerX 52
6.4.2.4. Agent接入（调度任务） 55
6.4.2.5. 在本地接入公网环境 55
6.4.2.6. 容器服务Kubernetes版接入SchedulerX 59
6.4.2.7. 在Kubernetes集群中部署SchedulerX 67
6.4.2.8. Endpoint列表 83
6.5. Scheduling和Triggers 84
6.5.1. 定时调度 84
6.5.1.1. Cron 84
6.5.1.2. Fixed rate 88
6.5.1.3. Second delay 89
6.5.1.4. One Time 91
6.5.2. 工作流调度 93
6.5.3. API 触发 95
6.5.4. 如何重刷数据 95
6.6. 任务类型 97
6.6.1. Java任务 97
6.6.2. 脚本任务 100
6.6.3. HTTP任务（Serverless） 101
6.6.4. DataWorks任务 108
6.6.5. XxlJob任务 109
6.7. 分布式编程模型 117
6.7.1. 单机 117
6.7.2. 广播 117

6.7.3. Map模型 119
6.7.4. MapReduce模型 124
6.7.5. 多语言版本分片模型 130
6.8. 高级特性 133
6.8.1. 如何接入日志服务 133
6.8.2. 如何使单应用支持十万以上的定时任务 142
6.8.3. 如何管理应用级别的资源和任务优先级 143
6.8.4. 如何创建秒级调度任务 146
6.8.5. 如何通过工作流进行上下游数据传递 147
6.8.6. 如何设置数据时间 149
6.8.7. 如何重刷数据 150
6.9. 控制台使用指南 152
6.9.1. 执行列表 152
6.9.2. 任务管理 153
6.9.3. 流程管理 159
6.9.4. 应用管理 161
6.9.5. 操作记录 165
6.10. API参考 166
6.10.1. API概览 166
6.10.2. 调用方式 166
6.10.3. 获取AccessKey 168
6.10.4. BatchDeleteJobs 170
6.10.5. BatchDisableJobs 172
6.10.6. BatchEnableJobs 174
6.10.7. CreateAppGroup 177
6.10.8. CreateJob 180
6.10.9. DeleteJob 186
6.10.10. DeleteWorkflow 189

6.10.11. DisableJob 190
6.10.12. DisableWorkflow 192
6.10.13. EnableJob 193
6.10.14. EnableWorkflow 196
6.10.15. ExecuteJob 197
6.10.16. ExecuteWorkflow 199
6.10.17. GetJobInfo 200
6.10.18. GetJobInstance 206
6.10.19. GetJobInstanceList 210
6.10.20. ListJobs 213
6.10.21. ListGroups 221
6.10.22. ListNamespaces 223
6.10.23. StopInstance 226
6.10.24. UpdateJob 228
6.11. 配置参考 232
6.11.1. JobContext参数说明 232
6.11.2. 任务管理高级配置参数说明 233
6.11.3. SchedulerxWorker配置参数说明 234
6.12. 常见问题 235
6.12.1. 从分布式任务调度1.0迁移到2.0 235
6.12.2. 常见问题 240
6.12.3. 报警常见问题 249
7.应用平台管理 250
7.1. 云服务集成 250
7.1.1. 云服务集成概述 250
7.1.2. 组件中心 251
7.1.2.1. 组件中心简介 251
7.1.2.2. 云服务总线 252

7.1.2.2.1. 云服务总线CSB简介 252
7.1.2.2.2. 管理服务开放实例 253
7.1.2.3. 批量运维 254
7.2. 权限管理 255
7.2.1. 权限管理概述 255
7.2.2. 账号体系 258
7.2.3. 将EDAS内置的权限管理切换为RAM权限管理 260
7.2.4. RAM权限管理 265
7.2.4.1. RAM简介 265
7.2.4.2. 管理RAM用户 265
7.2.4.3. 借助RAM角色实现跨云账号访问资源 266
7.2.4.4. 借助资源组进行权限管理 268
7.2.4.5. 列表鉴权 270
7.2.4.6. 使用标签控制资源访问 272
7.2.4.7. 在RAM中配置服务测试相关权限 275
7.2.4.8. 如何自动为ECS实例添加访问ACM所需的RAM角色 276
7.2.4.9. 权限策略示例库 279
7.2.5. 管理EDAS内置权限（不推荐） 292
7.2.6. 使用EDAS权限助手生成权限策略 294
7.3. 标签管理 305
7.3.1. 标签概述 305
7.3.2. 使用标签查找资源 305
7.3.3. 使用标签控制资源访问 306
7.4. 配置管理 310
7.4.1. 配置模板管理 310
7.4.2. 微服务配置 311
7.4.2.1. 配置管理概述 311
7.4.2.2. 配置管理的基本概念 312

7.4.2.3. 创建配置 313
7.4.2.4. 同步配置 315
7.4.2.5. 管理配置 317
7.4.2.6. 查看历史版本和回滚配置 319
7.4.2.7. 监听查询 319
7.4.2.8. 查询配置推送轨迹 320
7.4.3. Kubernetes配置 320
7.4.3.1. 管理配置项 320
7.4.3.2. 管理保密字典 324
7.5. 查看操作日志 327
7.6. 系统管理常见问题 328
7.6.1. 如何设置RAM用户的联系方式? 328
8.SDK参考 329
8.1. Java SDK接入指南 329
8.2. Python SDK接入指南 331
8.3. CLI接入指南 333
9.联系我们 334

1.金融云用户指南概述

EDAS已支持金融云。您可以在金融云中使用EDAS托管微服务应用、并进行微服务治理。

金融云简介

金融云是服务于银行、证券、保险、基金等金融机构的行业云，采用独立的机房集群提供满足一行两会监管要求的云产品，并为金融客户提供更加专业周到的服务。金融云按照人民银行和银监会的合规标准建设，在安全性、服务可用性和数据可靠性等方面作了大幅增强。更多信息，请参见金融云。

支持的金融云地域

EDAS支持杭州（cn-hangzhou-finance）、深圳（cn-shenzhen-finance-1）和上海（cn-shanghai-finance-1）3个金融云地域。在这3个地域，还包含一系列其他云产品，更多信息，请参见金融云产品列表。

使用限制

目前EDAS在金融云中提供的功能和公共云有些差异，导致在使用EDAS时会有以下限制，对您造成的不便，请谅解。

- 暂不支持杭州地域的K8s集群。
- 暂不支持1.22及以上版本的K8s集群。
- 暂不支持限流降级功能。
- 暂不支持使用代购的ECS集群进行应用管理，例如创建应用、应用扩容、弹性伸缩等场景。
- 暂不支持Serverless K8s集群。
- 暂不支持多语言应用。
- 暂不支持ECS应用的限流降级能力。
- 暂不支持组件中心的概览、批量运维、分布式任务调度能力。
- 暂不支持微服务治理模块的Dubbo Admin、标签路由、服务测试、自动化回归、服务降级、服务网格、全链路流量控制等能力。更多信息，请参见功能列表。

另外，在金融云中使用EDAS时需要依赖的其他云资源也有些限制，更多信息，请参见金融云产品限制。

应用托管

EDAS在金融云支持一系列应用托管的能力。

功能类型	功能描述	相关文档
资源管理	微服务空间 提供互相隔离的运行环境，如开发、测试和生产环境等。您可以使用命名空间实现资源和服务的隔离，并可以使用一个账号进行统一管理。	使用控制台管理微服务空间
	ECS集群管理 创建、扩容集群，以及其它集群管理操作，代购的ECS集群暂不支持集群管理功能。	• 管理ECS集群 • 使用控制台扩容ECS集群
	K8s集群管理 导入在容器服务Kubernetes版中创建的各种类型的K8s集群，暂不支持Serverless K8s集群。	• 创建和部署应用概述（K8s） • 升级和回滚应用概述（K8s）
	混合云 EDAS支持混合云ECS集群，且可以对混合云集群的扩容、网络和统一管理等问题提供完整的解决方案。将金融云ECS、本地IDC或及其它云服务提供商的服务器通过专线连通，并添加到EDAS混合云ECS集群中，即可将应用托管到混合云ECS集群，使用EDAS提供的一系列应用托管能力。	创建混合云ECS集群

功能类型	功能描述	相关文档
	资源组 当您使用云账号负责购买资源，或RAM用户负责应用运维时，可以使用资源组对账号的权限进行控制。EDAS可以对RAM用户进行资源组授权，被授权的RAM用户拥有操作这个资源组中所有资源的权限。	管理资源组
部署应用	• 创建和部署（首次部署）：在应用开发、测试完成后，可以在ECS集群中创建并部署。 • 升级和回滚：应用在EDAS中创建并部署后，还会不断迭代，需要升级。如果升级的应用版本发现问题，需要将应用回滚到历史版本。	• 创建和部署应用概述（K8s） • 应用创建和部署概述（ECS） • 升级和回滚应用概述（K8s） • 升级和回滚应用概述（ECS）
CI/CD	应用在不断迭代过程中，需要持续集成（CI）和持续部署（CD）。EDAS支持通过Jenkins和云效对您部署的应用进行CI/CD。	• CI/CD概述（K8s） • 使用云效2020部署Java应用至ECS集群
应用运维	EDAS为应用提供了一些列运维功能，包括： • 应用生命周期管理 • 负载均衡 • 变更记录 • 日志	• ECS 集群中的应用生命周期管理 • 负载均衡概述 • 查看应用变更 • 日志简介

服务治理

EDAS在金融云支持一系列服务治理的能力。

功能	描述	相关文档
调用链查询	• 通过设置查询条件，可以准确找出哪些业务性能较差，甚至异常。 • 基于调用链查询的结果，查看慢业务或出错业务的调用链的详细信息，进行依赖梳理，包括识别易故障点、性能瓶颈、强依赖等问题；也可以根据链路调用比例、峰值QPS评估容量。	• 查询调用链 • 查看调用链详情
服务拓扑	通过拓扑图的形式直观的了解不同服务间的相互调用关系及相关性能数据。	查看服务拓扑
服务鉴权	支持使用服务鉴权实现Spring Cloud、Dubbo应用的访问控制。当您的某个微服务应用有安全要求，不希望其它所有应用都能调用时，可以对调用该应用的其它应用进行鉴权，当前仅允许匹配鉴权规则的应用调用。	• 使用服务鉴权实现Spring Cloud应用的访问控制 • 使用服务鉴权实现Dubbo应用的访问控制
离群实例摘除	微服务架构中，当服务提供者的应用实例出现异常，而服务消费者无法感知时会影响服务的正常调用，并影响消费者的服务性能甚至可用性。离群实例摘除功能会检测应用实例的可用性并进行动态调整，以保证服务成功调用，从而提升业务的稳定性和服务质量。	• 使用离群实例摘除保障Spring Cloud应用的可用性 • 使用离群实例摘除保障Dubbo应用的可用性

功能列表

下表列出了金融云支持的功能列表，同时还展示了不同地域的功能对比，标识Y代表支持，N代表不支持。

功能		华东1	华东2	华南1
计费模式	只支持包年包月，暂不支持按量付费。			
资源管理	ECS	Y	Y	Y
	VPC	N	N	N
	ECS集群	Y	Y	Y
	容器服务K8s集群	N（暂不支持杭州地域K8s集群）	Y	Y
	Serverless K8s集群	N	N	N
权限资源管理	RAM鉴权	Y	Y	Y
	RAM资源组	N	N	N
	TAG支持	N	N	N
	配额	N	N	N
应用管理	命名空间	Y	Y	Y
	应用列表	Y	Y	Y
ECS应用	基本信息	Y	Y	Y
	金丝雀发布	Y	Y	Y
	变更记录	Y	Y	Y
	日志管理	Y	Y	Y
	日志在线查看	Y	Y	Y
	分布式日志搜索(SLS)	Y	Y	Y
	应用监控(ARMS)	Y	Y	Y
	报警（ARMS）	Y	Y	Y
	限流降级(AHAS)	N	N	N
	限流降级（老版本）	N	N	N
	配置推送	Y	Y	Y
	服务列表	Y	Y	Y
	弹性伸缩	Y	Y	Y
	事件中心	Y	Y	Y
	容器版本	Y	Y	Y
	基本信息	N	Y	Y

功能		华东1	华东2	华南1
K8s应用	变更信息	N	Y	Y
	应用事件	N	Y	Y
	实时日志	N	Y	Y
	日志目录	N	Y	Y
	日志文件（SLS）	N	Y	Y
	应用监控（ARMS）	N	Y	Y
	限流降级（AHAS）	N	N	N
	限流降级（老版本）	N	N	N
	通知报警（ARMS）	N	Y	Y
	事件中心	N	Y	Y
	服务列表	N	Y	Y
	K8s Service 管理	N	Y	Y
	弹性伸缩	N	Y	Y
多语言应用	K8s支持多语言应用	N	N	N
应用路由	K8s Ingress	Y	Y	Y
配置管理	配置列表历史版本监听查询推送轨迹	Y	Y	Y
	K8s配置	Y	Y	Y
微服务管理（治理）	调用链查询	Y	Y	Y
	服务查询	Y	Y	Y
	集群实例摘除	Y	Y	Y
	服务鉴权	Y	Y	Y
	Dubbo Admin	N	N	N
	标签路由	N	N	N
	服务测试	N	N	N
	服务压测	N	N	N
	自动化回归	N	N	N
	服务巡检	N	N	N
	服务降级	N	N	N

功能		华东1	华东2	华南1
	服务网格	N	N	N
	全链路流量控制	N	N	N
	组件中心	组件概览	N	N
		批量运维	N	N
分布式任务调度2.0		N	N	N
系统管理	主账号	Y	Y	Y
	子账号	Y	Y	Y
	角色	Y	Y	Y
	所有权限	Y	Y	Y
	权限助手	Y	Y	Y
	个人资料	Y	Y	Y
	账号切换	Y	Y	Y
	产品用量	Y	Y	Y
	操作日志	Y	Y	Y

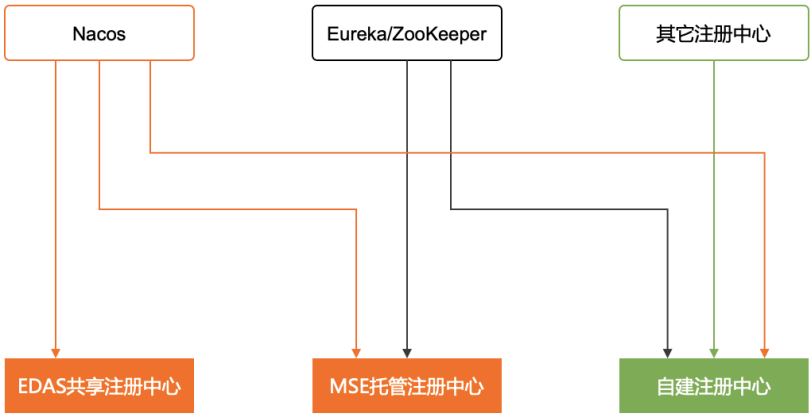
2.选择注册中心运维方式

Java微服务架构中常见的注册中心包含Eureka、ZooKeeper和Nacos等，用于实现服务的注册与发现，能够屏蔽、解耦服务之间的相互依赖，以便对微服务进行动态管理。本文介绍EDAS支持的注册中心、运维方式及相关操作。

注册中心

注册中心包含Eureka、ZooKeeper、Nacos等。关于各注册中心的更多信息，请参见各自官网或Github，帮助您根据实际需求选择。

无论您的应用使用哪种注册中心，EDAS都能够为您的应用提供托管和微服务治理能力。



从运维角度考虑，您在EDAS中部署的应用，使用不同类型的注册中心，可以选择不同的运维方式。

使用Nacos

如果使用Nacos作为注册中心，会有以下2个运维选择：

- 使用EDAS共享注册中心
  EDAS共享注册中心，即EDAS集成了Nacos的商用版本，以平台自带服务组件（免运维）的形式提供Nacos的服务注册与发现的能力。
  优势
  EDAS共享注册中心具有以下优势：
  - 共享组件，节省了部署、运维注册中心的成本。
  - 在服务注册和发现的调用中都进行了链路加密，保护您的服务，无需再担心服务被未授权的应用发现。
  - EDAS共享注册中心与EDAS其他组件紧密结合，为您提供一整套的微服务解决方案。

  操作
  在创建或部署应用时，取消选中**使用程序配置的注册中心**，则无论应用中如何配置Nacos的地址，都会被覆盖、连接到EDAS共享注册中心。
- 使用应用程序配置的注册中心，包含MSE托管和自建的Nacos**操作**
  在创建或部署应用时，选中**使用程序配置的注册中心**。则EDAS会为应用配置如下参数，以免应用部署时，配置的Nacos地址被替换为EDAS共享注册中心地址，从而保证您可以继续使用应用中配置的注册中心。

  > **说明** 保证您的注册中心地址与托管到EDAS中的应用网络互通，例如在同一个VPC内。

  ```
  -Dnacos.use.endpoint.parsing.rule=false
  -Dnacos.use.cloud.namespace.parsing=false
  ```

使用Eureka或ZooKeeper

如果使用Eureka或ZooKeeper，可以选择使用MSE托管或自建。MSE的更多信息及托管注册中心的优势，请参见什么是微服务引擎MSE。

> **说明** 保证您的注册中心地址与托管到EDAS中的应用网络互通，例如在同一个VPC内。

操作

（可选）使用Eureka或ZooKeeper，在EDAS创建或部署应用时，选中**使用程序配置的注册中心**。

您也可以在应用程序的配置中将Eureka或ZooKeeper更改为Nacos，以便使用EDAS共享注册中心。

使用其它类型的注册中心

如果使用其它类型的注册中心，例如Consul，只能继续使用您自建的注册中心。

> **说明** 保证您的注册中心地址与托管到EDAS中的应用网络互通，例如在同一个VPC内。

操作

（可选）使用其它类型的注册中心，在EDAS创建或部署应用时，需要选中**使用程序配置的注册中心**。

您也可以在应用程序的配置中将其它类型的注册中心更改为Nacos，以便使用EDAS共享注册中心。

3.资源检索

EDAS为您提供了目标资源检索能力，通过已掌握的资源信息ID、名称等实现对目标资源的快速查询和访问。

背景信息

EDAS中微服务空间、集群和应用，这三者每一类资源下都有若干个子资源或属性。当您需要通过子资源或属性反查所属父资源，例如当您根据Pod实例IP来查找该Pod所在的应用，或者需要找出某ECS所在的ECS集群时等等，往往很难实现。在实际应用场景中，这些情况很常见。因此，EDAS提供了便捷的资源检索功能来帮助您快速触达目标资源。

> **说明** 进行资源检索需要您具备资源搜索权限。授权资源检索的操作，请参见搜索权限控制。主账号无需授权即可使用。

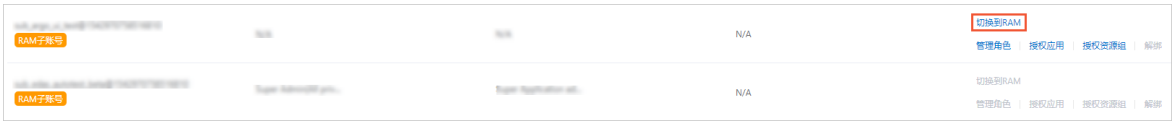
资源检索覆盖的具体范围如下：

- 微服务空间。
- 集群：包括ECS集群、容器服务K8s集群、Serverless K8s集群。
- 应用：包括在ECS集群、ACK集群和ASK集群中部署的应用。

搜索权限控制

资源搜索可支持对EDAS上微服务空间、集群和应用进行搜索，为保护数据安全，需要对账号进行资源搜索权限配置。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 子账号**。
3. 在**子账号**页面目标子账号的**RAM鉴权**列单击**切换到RAM**。



> **说明**
> - 已经使用了RAM授权的子账号，**操作**列的按钮将会无法单击。
> - 仍使用EDAS内置授权的子账号，可以选择切换到RAM授权，切换后将无法重新使用EDAS内置授权。

切换到RAM授权时，EDAS会预先判断该子账号是否已经在RAM控制台授予了EDAS的权限。

- 如果已经完成了RAM授权，在弹窗中单击**确定**，即可成功切换为RAM鉴权。
- 若子账号没有在RAM控制台被授予EDAS权限，将提示前往RAM控制台完成授权。

4. 在左侧导航栏选择**系统管理 > 权限助手**。
5. 在**权限助手**页面单击**创建权限策略**。
6. 在**创建权限策略**配置向导的**创建自定义权限策略**页签中设置权限策略的**策略名称**和**备注**。
7. 添加权限语句。

> **说明**
> - 在新增权限语句时，只能选择一种类型（允许或拒绝）的权限效力。
> - 您可以为一个权限策略创建多个权限语句，当某个权限的权限效力在不同权限语句中被设置为**允许**和**拒绝**时，遵循拒绝优先原则。

i. 在**创建自定义权限策略**页签中单击**新增权限语句**，在**加授权语句**面板中，将查看微服务空间test、查看微服务空间test下的所有集群，以及操作微服务空间test下所有应用的**权限效力**设置为**允许**，然后单击**确认**。

a. 在**权限效力**下方选择**允许**。

b. 在**操作与资源授权**左侧的权限列表中选择**命名空间 > 查看命名空间**，在右侧的资源列表中选择*华北2（北京）*和*test*。

c. 在**操作与资源授权**左侧的权限列表中选择**集群 > 查看集群**，在右侧的资源列表中选择*华北2（北京）*、*test*和*全部集群*。

d. 在**操作与资源授权**左侧的权限列表中选择**应用**（该操作会选中应用下的所有权限），在右侧的资源列表中选择*华北2（北京）*和*test*。

ii. 在**创建自定义权限策略**页签中单击**新增权限语句**，在**加授权语句**面板中，将微服务空间test下创建应用的**权限效力**设置为**拒绝**，然后单击**确认**。

a. 在**权限效力**下方选择**拒绝**。

b. 在**操作与资源授权**左侧的权限列表中选择**应用 > 创建应用**，在右侧的资源列表中选择*华北2（北京）*和*test*。

8. 在**策略预览**页签预览权限，然后单击**完成**。
控制台面板将会提示 `新增策略授权成功` ，单击**返回列表查看**可返回**权限助手**页面，查看新建的权限策略模板。
9. 在**策略预览**页签根据页面提示在RAM控制台创建自定义权限策略，并请参见步骤三：创建RAM用户并添加授权授权给对应的RAM用户。

搜索资源

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**资源搜索（CTRL + F）**，在弹出的资源搜索框中输入待查询的资源关键字。

如果您掌握待查询关键字的资源属性，可通过属性前缀进行细粒度查询，当前支持的属性前缀如下：

- `id` ：资源的唯一标识；
- `name` ：资源的名称；

- `ip`：资源的IP地址；
- `tag`：资源的标签。

搜索框还支持如下两个帮助指令：

- **help**：提供帮助信息；
- **info**：提供当前登录用户的信息。

> **说明** 当前EDAS控制台除K8s应用详情页外，其余页面均可通过在键盘敲击*CTRL+F*键的方式来弹出资源搜索框实现即时查询。

3. 单击右侧资源ID进入资源详情页查看。

4.应用模板管理

您可使用EDAS应用模板功能将同一份应用配置部署到多个应用，构建应用在多个Kubernetes集群发布管理的能力，同时提供相关应用持续运维的能力，并可以发布应用。

前提条件

- 您的阿里云账号已同时开通EDAS和容器服务ACK，请参见：
  - 开通EDAS
  - 首次使用容器服务ACK
- 在容器服务ACK完成角色授权，请参见容器服务默认角色。
- 创建容器服务ACK集群
- 在EDAS控制台中导入容器服务K8s集群

创建应用模板

1. 登录EDAS控制台。
2. 在左侧导航栏中，单击**应用模板**，在**应用模板**页面，单击**创建模板**。
3. 在**模板基本信息**配置向导页面，选择应用运行环境，然后单击**下一步**。

参数	描述
选择应用	应用的运行环境。不同部署包类型，需要选择不同的Java应用环境。 ○ **自定义**：使用镜像部署Java应用。应用运行环境包含在镜像中，无需选择。 ○ **Java**：支持通用的JAR包部署，适用于Dubbo和Spring Boot应用。选择后，可设置**Java环境**。 ○ **Tomcat**：支持通用的WAR包部署，适用于Dubbo和Spring应用。选择后，可设置**Java环境**和**容器版本**。 ○ **EDAS-Container（HSF）**：适用于使用WAR或者FatJar部署HSF应用。选择后，可设置**Java环境**、**Pandora版本**和**Ali-Tomcat版本**。
服务注册与发现配置	选择注册中心的运维方式。如何选择，请参见选择注册中心运维方式。
监控及治理方案	K8s集群默认自动挂载Java Agent进行精细化监控，并提供完整微服务治理方案（金丝雀发布、服务鉴权、限流降级等）。

4. 在**模板配置**配置向导页面，配置模板信息。
   - 自定义镜像部署应用模板

参数	描述
模板名称	输入模板名称，必须以小写字母开头，允许小写字母、数字以及中划线组合。最大长度63个字符。
（可选）模板描述	输入模板描述 ，最大长度为128个字符。

参数	描述
选择镜像	■ **阿里云镜像服务**选择**当前账号**。 选择**镜像所属地域**、**容器镜像服务**、**镜像仓库命名空间**、**镜像仓库名**和镜像版本。 ■ **阿里云镜像服务**选择为**其他阿里云账号**。 ■ 如您的镜像存放在公开仓库中，那么您配置**完整镜像地址**即可。 ■ 如您的镜像存放在私有仓库中，那么您需要使用免密插件拉取容器镜像，请参见使用免密组件拉取容器镜像。 **说明** 如果您以RAM用户且使用企业版容器镜像仓库里的镜像创建应用时，需要阿里云账号为该RAM用户授权。具体信息，请参见配置仓库的RAM访问控制。
Pod总数	设置该应用要部署的Pod个数。
单Pod资源配额	设置单个Pod的CPU和内存，如果您需要限额，请填写具体的数字，使用默认值0则代表不限额。

- JAR包、WAR包或部署包部署应用模板

参数	描述
模板名称	输入模板名称，必须以小写字母开头，允许小写字母、数字以及中划线组合。最大长度63个字符
（可选）模板描述	输入模板描述 ，最大长度为128个字符。
文件上传方式	选择包的上传方式。 ■ **选择应用**为**Java**时，上传配置如下： ■ **上传JAR包**：选择下载好的JAR包并上传。 ■ **JAR包地址**：输入JAR包地址。 ■ **选择应用**为**Tomcat**时，上传配置如下： ■ **上传WAR包**：选择下载好的WAR包并上传。 ■ **WAR包地址**：输入WAR包地址。 ■ **选择应用**为**EDAS-Container（HSF）**时，上传配置如下： ■ **上传部署包**：选择下载好的部署包并上传。 ■ **部署包地址**：输入部署包地址。
版本	请输入版本，您可以自定义版本号，也可以单击右侧的**用时间戳作为版本号**自动生成版本号。
时区	设置应用的时区信息。**选择应用**为**EDAS-Container（HSF）**时，无此配置项。
Pod总数	设置该应用要部署的Pod个数。

参数	描述
（可选）单Pod资源配额	设置单个Pod的CPU和内存，如果您需要限额，请填写具体的数字，使用默认值0则代表不限额。

5. 单击**下一步**，在**模板高级配置**配置向导页面，配置应用调度规则、启动命令、环境变量等，配置完成后单击**下一步**，在**模板创建完成**配置向导页面，确认模板配置信息，然后单击**确认创建模板**。

   模板高级配置的具体配置说明，请参见：

   - 配置调度规则
   - 配置启动命令
   - 配置环境变量
   - 配置持久化存储
   - 配置本地存储
   - 配置应用生命周期的钩子和探针
   - 配置日志收集
   - 配置Tomcat
   - 配置Java启动参数
   - 实现K8s集群应用的限流降级
   - 配置挂载

   如无需进行模板的高级配置，则可直接在**模板配置**配置向导页面，单击**创建模板**。

6. 在**提示**对话框，单击**确定**。
   在**应用模板**页面可以看到您刚创建成功的模板。

更多操作

在**应用模板**页面，您可以根据需要执行如下操作：

- 查看模板详情：在目标应用模板的**操作**列，单击**详情**。
- 使用模板创建应用：在目标应用模板的**操作**列，单击**创建应用**。更多信息，请参见创建多集群应用。
- 修改应用模板配置：在目标应用模板的**操作**列，单击**编辑**。更多信息，请参见模板配置和模板高级配置。

  > **说明**
  > - 修改应用模版不会导致使用了此模板的应用立即更新，需要等到下次对应的多集群应用部署。
  > - 如果在多集群应用中修改了部分配置，即使在应用模板中也修改了相同配置，该部分配置在应用中也不会生效。

- 删除应用模板：在目标应用模板的**操作**列，单击**删除**。

  > **说明** 删除应用模板前，需先删除使用此模板的所有多集群应用。

5.多集群应用管理

多集群应用管理模块实现了EDAS多容器服务K8s集群应用生命周期的管理，通过应用模板在多集群内部署应用，支持下发的各单集群应用在应用模板配置基础上自定义配置，使配置下发方式更加灵活，支持多种部署方式（单批和分批），同时在发布操作后提供变更记录的查询功能。

使用场景

- 将一份应用配置部署到多个集群实现容灾或负载均衡功能。
- 将一份应用配置在同一个集群上部署多份实例实现灰度、AB测试等功能。
- 将一份应用配置部署到多个地域实现容灾或全球化等。

注意事项

使用多集群应用创建的应用，不建议单独在**应用列表**页面单独部署应用，否则下一次部署多集群应用时，会覆盖掉之前的修改内容。

前提条件

创建应用模板

创建多集群应用

1. 登录EDAS控制台。
2. 在左侧导航栏中，单击**多集群应用**。
3. 在**多集群应用**页面，单击**创建多集群应用**。
4. 在**应用配置**的配置向导页面，填写多集群应用名称和描述，选择应用模板，然后单击**下一步**。

   如果您没有创建应用模板，或者需要创建新的应用模板，您可以直接单击**创建应用模板**，创建一个全新的应用模板。具体操作，请参见应用模板管理。
5. 在**发布配置**的配置向导页面，添加目标集群和创建目标应用，然后单击**开始发布**。
   i. 单击**添加集群**，在**添加集群**对话框中，选择地域并勾选目标集群，然后单击**确认**。
   ii. 在单目标集群页签，修改应用配置。
   如果需要添加应用，您可以单击**创建目标应用**，然后配置添加的应用。

参数	描述
微服务空间	应用所属空间。选择您创建的微服务空间，如果您未创建微服务空间或不做选择，微服务空间则设置为**默认**。 如果您没有创建微服务空间，或者需要创建新的微服务空间，您可以直接单击**创建微服务空间**，创建一个全新的微服务空间。具体操作，请参见使用控制台管理微服务空间。
集群	上一步选择的容器服务K8s集群。
K8s Namespace	K8s Namespace通过将系统内部的对象分配到不同的Namespace中，形成逻辑上分组的不同项目、小组或用户组，便于不同的分组在共享使用整个集群的资源的同时还能被分别管理。 ■ **default**：没有其他命名空间的对象的默认命名空间。 ■ **kube-system**：系统创建的对象的命名空间。 ■ **kube-public**：此命名空间是自动创建的，并且可供所有用户（包括未经过身份验证的用户）读取。 ■ 其他为自定义创建的K8s Namespace。

参数	描述
应用名称	输入应用名称，必须以字母开头，允许数字、字母、短划线（-）组合。最多可输入36个字符。
应用描述	输入应用描述 ，最大长度为128个字符。
镜像部署应用	
应用部署方式	部署应用使用的镜像。默认使用应用模板中选择的镜像。 单击，在**选择镜像**对话框，选择需要使用的镜像，然后单击**确定**。 ■ **阿里云镜像服务**选择**当前账号**。 选择**镜像所属地域**、**容器镜像服务**、**镜像仓库命名空间**、**镜像仓库名**和镜像版本。 ■ **阿里云镜像服务**选择为**其他阿里云账号**。 ■ 如您的镜像存放在公开仓库中，那么您配置**完整镜像地址**即可。 ■ 如您的镜像存放在私有仓库中，那么您需要使用免密插件拉取容器镜像，请参见使用免密组件拉取容器镜像。
JAR包部署应用	
Java环境	部署应用运行的环境版本。默认使用应用模板中选择的版本。
上传方式	选择JAR包上传方式。 ■ **上传JAR包**：选择下载好的JAR包并上传。 ■ **JAR包地址**：输入Demo包地址。
WAR包部署应用	
Tomcat环境	部署应用运行的环境版本。默认使用应用模板中选择的版本。
Tomcat环境	选择WAR包上传方式。 ■ **上传WAR包**：选择下载好的WAR包并上传。 ■ **WAR包地址**：输入Demo包地址。
WAR或者FatJar部署HSF应用	
EDAS-Container（HSF）环境	部署应用运行的环境版本。默认使用应用模板中选择的版本。
Tomcat环境	■ **上传部署包**：选择下载好的部署包并上传。 ■ **部署包地址**：输入Demo包地址。
通用配置参数	
Pod总数	设置该应用要部署的Pod个数。

参数	描述
单Pod资源配额	设置单个Pod的CPU和内存，如果您需要限额，请填写具体的数字，使用默认值0则代表不限额。
服务注册与发现配置	选择注册中心的运维方式。如何选择，请参见选择注册中心运维方式。
高级配置	您可以根据需要配置应用调度规则、启动命令和环境变量等。具体参数说明，请参见： ■ 配置调度规则 ■ 配置启动命令 ■ 配置环境变量 ■ 配置持久化存储 ■ 配置本地存储 ■ 配置应用生命周期的钩子和探针 ■ 配置日志收集 ■ 配置Tomcat ■ 配置Java启动参数 ■ 实现K8s集群应用的限流降级 ■ 配置挂载

6. 在**确认发布**对话框中，确认配置信息，然后单击**确定**。
7. 在**变更记录**页面，查看应用的变更状态。

   状态为**已完成**时，应用创建成功。
   - 在**多集群应用**页面可以看到您刚创建成功的多集群应用。在**操作**列，您可以进行以下操作：
     - 单击**详情**，您可以查看多集群应用的详情。
     - 单击**删除**，你可以删除多集群应用。

       > **说明** 删除多集群应用前，需先删除多集群应用下的所有应用。
   - 在**应用列表**页面可以看到您刚创建多集群应用时创建的目标应用。

部署多集群应用

1. 在**多集群应用**页面，找到目标多集群应用，在其**操作**列，单击**部署**。
2. 在**部署配置**的配置向导页面，配置应用，然后单击**下一步**。

   应用配置的具体参数说明，请参见发布配置。
3. 在**选择部署方式**的配置向导页面，在单批、分批或金丝雀发布区域，单击**开始部署**。
4. 在**发布确认**对话框，单击**确认**。
5. 在**变更记录**页面，查看应用的变更状态。

   状态为**已完成**时，应用部署成功。

6.分布式任务调度

6.1. 动态与公告

6.1.1. 功能发布记录

6.1.1.1. 2022年

本文介绍了SchedulerX产品在2022年发布涉及的新增功能、功能优化、重要问题修复及对应的文档，帮助您了解SchedulerX的发布动态。

客户端发布记录

1.5.1.1，2022-04-19

功能名称	功能描述	变更类型	相关文档
日志服务功能更新	支持开启或关闭日志服务动态生效。	新增	无
问题修复	• 修复1.5.0.x版本不兼容Spring Boot 2.0.3的问题。 • 修复无需用户配置JobSyncService Bean的初始化问题。	优化	无

1.5.0.2，2022-04-01

功能名称	功能描述	变更类型	相关文档
SpringBoot声明式任务定义	支持SpringBoot声明式任务定义。命名空间、应用、任务、报警等都可以通过配置文件声明，方便管理应用下的任务。文件声明支持修改，您可以配置在任何环境一键启动应用。	新增	Spring Cloud Alibaba定时任务
可视化MapReduce任务	可视化MapReduce任务（原并行计算），支持子任务级别列表、日志、重跑；支持为子任务自定义标签并且通过标签搜索子任务。	新增	企业级分布式批处理方案

1.4.2，2022-03-07

功能名称	功能描述	变更类型	相关文档
支持日志服务	分布式任务调度系统SchedulerX 2.0的日志服务，您不需要修改一行代码，只需要增加一个Log4j或Logback的配置，即可在控制台看到每次任务调度（包括分布式任务）的业务日志。	新增	如何接入日志服务
支持查看堆栈	控制台可以直接查看任务运行的堆栈，方便排查任务卡住的问题，使用时需要您将客户端升级到1.4.0及以上版本。	新增	无
支持ElasticJob	支持ElasticJob开发的执行器对接。	新增	兼容ElasticJob

1.3.4，2022-1-6

功能名称	功能描述	变更类型	相关文档
支持的任务类型	• 支持DataWorks任务。 • 新增 `schedulerx2-plugin-xxljob` ，兼容Xxl-job接口。	新增	• DataWorks任务 • XxlJob任务

服务端发布记录

2022-04-19

功能名称	功能描述	变更类型	相关文档
专业版功能更新	• 支持批量启用和禁用任务。 • 支持分页、按时间查询业务日志，增加显示日志时间。 • HTTP任务执行超时时间最大支持至2分钟。	新增	• 任务管理 • 如何接入日志服务 • HTTP任务（Serverless）
应用ID限制	应用ID长度最大限制修改为64个字符。	优化	无
任务管理功能优化	• 优化任务执行模式描述信息。 • 优化各场景下任务强制停止的原因说明。 • 修复任务指定机器都掉线时，开启故障机器自动转移功能无效的问题。	优化	任务管理

2022-03-15

功能名称	功能描述	变更类型	相关文档
专业版功能更新	概览页功能更新： • 增加每分钟触发汇总图表。 • 增加汇总数据链接跳转。 • 增加在线Worker列表展示。	新增	无
专业版功能差异化	专业版和基础版进行功能差异化并优化专业版对应功能。	新增	专业版公测
并行计算功能升级	并行计算任务支持用户在切分子任务时自定义子任务标签，控制台子任务列表支持基于子任务查询。	新增	企业级分布式批处理方案
支持ElasticJob	支持ElasticJob开发的执行器对接。	新增	兼容ElasticJob

2022-01-26

功能名称	功能描述	变更类型	相关文档

功能名称	功能描述	变更类型	相关文档
全新专业版公测	• 支持概览页。 • 支持日志服务。 • 支持查看堆栈。控制台可以直接查看任务运行的堆栈，方便排查任务卡住的问题，使用时需要您将客户端升级到1.4.0及以上版本。	新增	如何接入日志服务

2022-01-05

功能名称	功能描述	变更类型	相关文档
支持的任务类型	• 支持DataWorks任务。 • 支持一次性任务。 • 支持Xxljob任务，兼容开源Xxl-Job接口。 • 支持任务导入导出。	新增	• DataWorks任务 • XxlJob任务 • One Time

6.1.1.2. 历史记录

本文介绍了SchedulerX产品在2022年以前发布涉及的新增功能、功能优化、重要问题修复及对应的文档，帮助您了解SchedulerX的历史发布动态。

服务端

2021-12-01

新特性

- 新增API：指定机器。
- 新增API：查询工作流。

2021-06-02

新特性

- 支持应用自动分裂，可以支持单应用百万级别任务。
- 支持RAM鉴权。

2020-12-07

新特性

- 客户端实例支持自定义标签，可以进行任务指定标签运行。
- 新增邮件、钉钉机器人报警。
- 支持应用级别报警联系人组。

问题修复

- 修复修改GroupID，历史执行记录会误降级为10条的问题。
- 修复客户端实例数很多的时候，子任务详情机器列表无法分页的问题。
- 修复任务管理查看历史记录无数据的问题。

2020-09-30

新特性

- 公网region支持pop API。
- 支持使用appKey鉴权（应用Key，代替AccessKey ID或AccessKey Secret）。
- HTTP任务支持高级配置。
- 创建应用和创建任务支持操作记录。

优化

- 优化14寸电脑执行列表操作显示不全的内容格式问题。
- 控制台接口增加鉴权，防止系统安全受到攻击。

问题修复

- 修复通过任务管理查看历史记录是空白的问题。
- 修复池子中任务可能会卡住的问题。
- 修复广播任务可能会卡住的问题。
- 修复工作流中的HTTP任务，手动运行一次会误报无可用机器的问题。

2020-08-19

新特性

- 支持一个应用10万+任务。
- 任务管理新增详情查看功能。

优化

- 搜索性能优化。
- 应用列表下拉框，小于10个有全部应用，超过10个默认选中第一个。
- 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，池子状态的实例不会变为终止状态问题。
- 修复数据库异常导致池子中的实例会一直卡住问题。
- 修复删除应用，再重新创建个同名的应用会失败问题。
- 修复用户重启后，秒级任务会停止调度问题。

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popAPI创建HTTP任务。
- 支持授权和取消权限的popAPI。
- 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界面显示问题。
- 修复服务端切换主机，指定机器会丢失的问题。

客户端

1.3.2，2021-12-15

修复问题：调整默认依赖的log4j2至2.15.0版本。

1.3.0.3，2021-11-26

新特性

- 秒级别单机增强：单机秒级别循环支持在不用的实例间分发切换执行，SpringBoot应用配置参数如下：`spring.schedulerx2.enableSecondDelayStandaloneDispatch=true`（默认`false`）。

问题修复

- 修复“客户端对接多个应用分组时运行实例心跳只上报给指定的调度服务”的问题。
- 修复“秒级别任务偶发的线程异常中断和空指针异常”的问题。
- 修复“SchedulerX客户端与Spring Cloud集成时会出现任务运行锁死超时”的兼容问题。
- 修复“广播任务在并发情况下出现执行中断”的问题。
- 修复“秒级别任务在执行过程中未能定时刷新Worker实例列表”的问题。
- 修复“秒级别广播任务执行停止指令后，扫描线程未结束”的问题。
- 修复“秒级别广播任务场景下，大批量Worker按批次发布过程中任务有概率卡住”的问题。

1.2.9.1，2021-8-30

问题修复

- 修复问题：修复worker与Spring Cloud集成时出现任务运行锁死超时。

1.2.9，2021-8-27

新特性

- 秒级别单机增强：单机秒级别循环支持在不用的实例间分发切换执行，配置参数：`spring.schedulerx2.enableSecondDelayStandaloneDispatch=true`（默认`false`）。

问题修复

- 修复问题：广播任务在并发情况下出现中断信号，以及worker可以实例列表未更新。

1.2.8.3，2021-8-13

问题修复

- 修复秒级别广播任务执行停止指令后扫描线程未结束。

1.2.8.2，2021-8-6

问题修复

- 秒级别广播任务大批量worker按批次发布过程中任务运行有概率卡住进行修复。

1.2.8，2021-6-23

新特性

- 广播任务增强：postProcess可以拿到所有机器执行的状态`JobContext.getTaskStatuses()`和执行结果`JobContext.getTaskResults()`。
- 广播分片模型：广播任务可以通过JobContext获取`shardingNum`和`sharingId`，通过自己机器的index进行数据分布式处理。

- 分片模型增强：JobContext可以从 `shardingNum` 获取所有分片数量。

1.2.7，2021-4-28

新特性

- 新增开关，可以关闭的failover功能。

问题修复

- 修复心跳探活失败，可能会socket泄漏的问题。
- 修复MapTask Master在子任务比较多的情况下，可能会频繁误failover的问题。

1.2.5.2，2021-4-8

新特性

- 秒级别任务间隔支持到毫秒级别。

优化

- MapReduce模型，root Task固定在master node执行，方便排查问题。
- 不强依赖log4j2，如果用户使用的是logback，可以把log4j和log4j2依赖移除。

1.2.4.3，2021-1-21

新特性

- 广播任务运行中支持进度汇报。
- 心跳日志可以通过开关关闭。

问题修复

- 修复广播任务有可能卡住的严重问题。

优化

- 心跳间隔调整为10秒。

1.2.3.1，2020-12-16

问题修复

修复客户端负载高把任务调度长连接打挂，导致任务无法继续调度的问题。

1.2.2.2，2020-12-10

新特性

- 支持容器内采集容器真实CPU使用率。需要增加配置，以starter为例：

  ```
  spring.schedulerx2.enableCgroupMetrics=true，spring.schedulerx2.cgroupPathPrefix=/sys/fs/cgroup/cpu/
  ```

  （非必填，如果容器里有" `/sys/fs/cgroup/cpu/` "这个路径，可以不用配置，否则修改为真实的cgroup路径）。

- 支持标签。客户端启动可以自定义打标签，以starter为例： `spring.schedulerx2.labels=xxx` 。任务管理中可以指定机器，可以指定任务跑在某个标签的实例上。适用场景灰度、单元化等。

问题修复

- 把h2从shade中移除。shade h2可能会导致MySQL驱动加载失败。

优化

- 客户端心跳线程和akka核心线程独立出来，保证业务繁忙不会影响心跳探活。

1.2.1.2，2020-10-20

新特性

- 支持共享ContainerPool，客户端所有任务可以共享同一个线程池，大量任务高并发调度情况下大大提高客户端性能和稳定性。
- Shade Scala，解决scala冲突的问题。
- 公有云支持appKey鉴权。

- MapReduce模型增强，支持配置是否所有子任务成功才执行Reduce方法。
- 客户端支持 `-Dschedulerx.appKey` 设置 `appKey` 。

问题修复

- 修复广播任务可能会卡住的问题。
- 修复sls功能的AccessKey ID或AccessKey Secret泄漏的问题。

优化

- 优化客户端netty线程池。

1.2.0.2，2020-08-19

新特性

- 支持一个应用10万+任务（只有公有云支持）。
- 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问题。
- EDAS应用无法读取AccessKey Secret。

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` 挂起会导致任务卡住。

1.1.2.RELEASE，2020-02-10

新特性

- `shade protobuf and netty from AccessKey IDka` ，解决接入90%以上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` ）:HTTPS://yq.aliyun.com/articles/739601。
- 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自身及下游。

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，如果没有配置Domain，可能还是会启动失败。

优化

- Server端性能优化，和客户端通信同步改成异步，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。

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

优化：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

> **说明**
> 该版本控制台显示版本号为1.0.4与Maven版本号不一致，请知晓。

新特性

- 新增部分包的 `shade: aliyun-log, commons-validator, gson, fastjson, guava, 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。

6.1.2. 产品公告

6.1.2.1. 专业版商业化公告（2022年05月24日）

阿里云分布式任务调度SchedulerX专业版将于2022年06月25日正式进行商业化。

公测期实例转商用说明

对于公测期间在分布式任务调度SchedulerX上创建的专业版应用，您需要根据实际需求进行处理。

- 继续使用：您无需进行任何操作。从2022年06月25日00:00:00开始，分布式任务调度SchedulerX依据按量付费模式对您公测期间所创建的专业版应用进行计费。公测期间的实例自动转换为商用实例，转换过程中不会影响您已有的调度任务。
- 不再使用：为了避免误扣费用，请于2022年06月25日00:00:00前将专业版应用切换回基础版应用，切换之后会按照基础版计费规则计费，同时您也将不再享有专业版的任务调度能力。

> **说明** SchedulerX专业版和基础版的能力对比及计费规则详情，请参见计费说明。

6.1.2.2. 专业版公测（2022年01月26日）

阿里巴巴分布式任务调度平台SchedulerX 2.0的专业版于2022年01月26正式公测，本次公测带来了全新的可视化功能，兼容开源XXL-JOB任务，支持一次性任务，融合大数据DataWorks任务。

可视化

日志服务

在当前微服务和容器化越来越流行的情况下，可视化日志工具逐渐成为了企业的刚需，它可以帮助您在机器数量太多、没有权限登录容器等典型场景下实现任务失败原因的快速定位。

使用SchedulerX 2.0的日志服务，您不需要修改一行代码，只需要增加一个Log4j或Logback的配置，即可将每次任务调度的框架日志和业务日志进行收集，同时SchedulerX 2.0的专业版提供了日志检索功能，可以通过任务调度平台快速定位任务失败的原因。

更多信息，请参见如何接入日志服务。

查看堆栈

业务执行慢或者卡住时，查看堆栈是最有效的手段，但是现在的应用往往都是分布式或多线程，如何快速定位是哪个机器哪个线程卡住，逐渐成为了业务人员的刚需。

SchedulerX 2.0提供了白屏的查看堆栈功能，可以自动定位到任务执行的机器和线程，白屏打印堆栈信息，快速定位任务卡住的原因。

用户大盘

新增概览界面不但提供产品快报、用户手册、工单入口等能力，还支持可视化的用户大盘功能。

用户大盘会统计并展示当前任务总数、接入的Worker数量、运行中的任务实例数等。

兼容开源XXL-JOB

SchedulerX 2.0兼容XXL-JOB任务接口，支持@XxlJob新注解和@JobHandler老注解方式，您不需要修改一行代码，即可以将XXL-JOB任务在SchedulerX 2.0平台上托管。同时SchedulerX 2.0的专业版提供了商业化的报警和可视化功能。

更多信息，请参见XxlJob任务。

支持一次性任务

SchedulerX 2.0支持一次性任务，可以在未来的某一时刻执行一次，任务自动销毁。
更多信息，请参见One Time。

融合大数据DataWorks任务

在实际业务场景中业务处理往往依赖前置数据准备，目前在分布式任务调度平台上可进行DataWorks任务数据处理与业务数据处理任务依赖编排定时调度。
更多信息，请参见DataWorks任务。

如何升级为专业版

操作步骤

1. 访问分布式任务调度平台。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏单击**应用管理**。
4. 单击目标应用操作列的**编辑**。
5. 在弹出的面板中找到**高级配置**并单击展开图标，修改**版本**为专业版。

6. 单击**下一步**，根据实际情况修改**定时配置**。相关参数说明，请参见任务管理。
7. 单击**下一步**，根据实际情况修改**报警配置**。相关参数说明，请参见任务管理。
8. 单击**完成**。

基础版和专业版的区别

功能	基础版	专业版
基础调度能力	支持	支持
日志服务	不支持	支持
查看堆栈	不支持	支持
用户大盘	不支持	支持

功能	基础版	专业版
任务导入导出	不支持	支持
工作流实例图	不支持	支持
可视化MapReduce（并行计算）	最大300个子任务，无搜索能力	最大1000个子任务，有搜索能力
历史记录	最近10条/任务	最近100条/任务
报警	钉钉、邮件	钉钉、邮件、短信、电话
单应用分组任务数	1000	10万（如需扩容请联系管理员）
API	仅支持创建、更新、删除任务的API	所有API都支持
计费说明	按任务托管费用计费 任务托管CU=启用任务数×接入的Worker数	公测期内和基础版的收费方式相同

6.1.2.3. 基础版商业化公告（2021年07月30日）

为提供更优质的服务，阿里云分布式任务调度SchedulerX将于2021年09月01日正式商业化。对于公测期间创建的实例，如果2021年09月01日00:00:00前没有释放，那么我们将默认您所创建的集群采用按量付费模式开始收费。

开通商用实例

分布式任务调度SchedulerX商业化之后，将会以商用方式提供服务，您可以直接开通商用版本。相关的计费项和价格，请参见计费说明。

公测期实例转正式商用说明

对于公测期间在分布式任务调度SchedulerX上创建的调度任务，请根据实际需求处理。

- 继续使用公测：
  您无需任何操作，从2021年09月01日00:00:00开始分布式任务调度SchedulerX依据按量付费模式对您公测期间所创建的所有实例进行计费。公测期间的实例自动转换为商用实例，转换过程中不会影响您已有的调度任务。
- 不再使用公测：
  为了避免误扣费，请于2021年09月01日00:00:00前删除或停用调度的任务。具体操作，请参见删除调度任务。

6.2. 产品简介

6.2.1. 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，请搜索群号*23103656*加入钉钉答疑群咨询。

6.2.2. 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

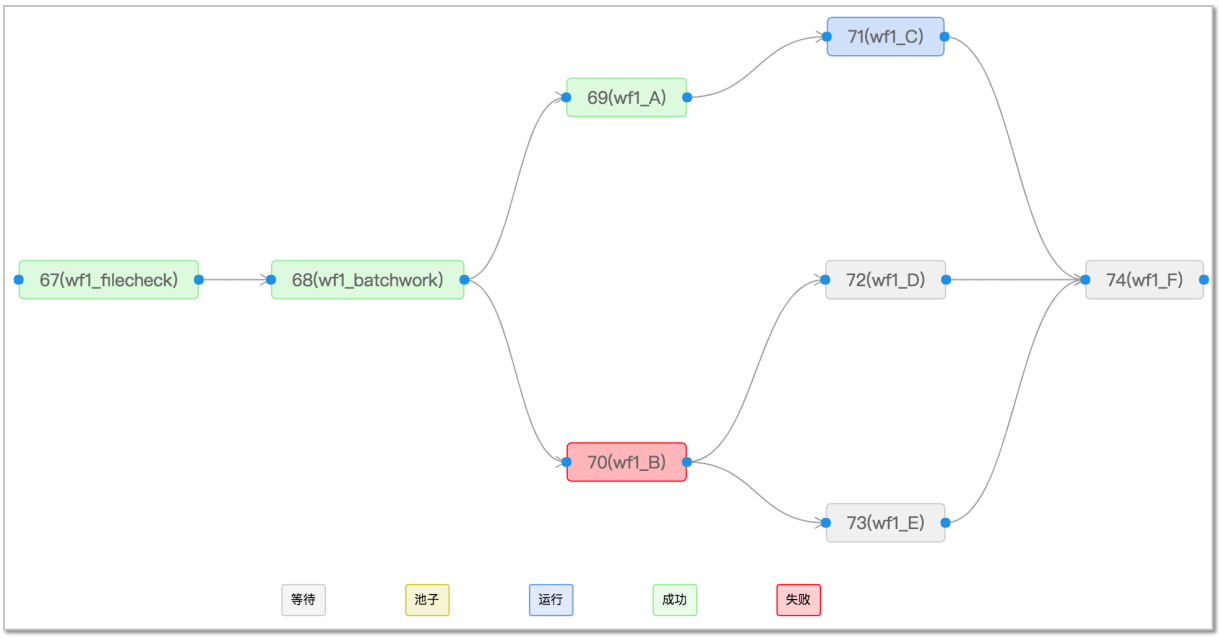
- Crontab：支持Unix Crontab表达式，详情请参见Cron。不支持秒级别。

- 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。

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

6.2.3. 产品优势

本文主要介绍SchedulerX 2.0为您提供的关键能力优势。

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用户和临时AccessKey ID。

6.2.4. 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）。

6.2.5. 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。

产品名称	定时调度	工作流	分布式任务	白屏化任务治理	任务类型	报警监控	使用成本

产品名称	定时调度	工作流	分布式任务	白屏化任务治理	任务类型	报警监控	使用成本
Quartz	Cron	不支持	不支持	不支持	Java	不支持	1个数据库、多个服务器、人工运维成本
Elastic-job	Cron	不支持	静态分片	• 支持运行大盘 • 支持运行日志 • 不支持原地重跑 • 不支持重刷数据 • 不支持执行记录	Java、Shell	自研	多个Zookeeper集群、人工运维成本
Xxl-Job	Cron	不支持	静态分片	• 支持执行记录 • 支持运行大盘 • 支持运行日志 • 不支持原地重跑 • 不支持重刷数据	Java、Shell、Python、PHP、Nodejs	邮件	1个数据库、1个调度中心、多个执行器、人工运维成本
SchedulerX 2.0	Cron、Fixed、Delay、Fixed Rate、One_Time、OpenAPI	支持，可以通过图形化配置，并且任务间可数据传递	静态分片、MapReduce动态分片	• 支持执行记录 • 支持运行大盘 • 支持运行日志 • 支持原地重跑 • 支持重刷数据	Java、Shell、Python、Go、HTTP、Nodejs、可自定义	短信、钉钉、邮件、电话	按照调度量和计算量收费，无机器和人工运维成本

6.3. 产品计费

6.3.1. 计费说明

本文介绍分布式任务调度SchedulerX的各计费项及其计费规则。

基本概念

- 任务类型：调度任务的类型。包含单机任务和非单机任务（广播、分片、并行和网络任务等）。
- 任务调度总量：通过SchedulerX配置并启用的分布式任务调度总规模，单位为CU（Capacity Unit）。

产品版本对比

SchedulerX目前支持基础版和专业版两个版本，各版本所支持的功能如下：

功能	基础版	专业版
基础调度能力	支持	支持
日志服务	不支持	支持
查看堆栈	不支持	支持
用户大盘	不支持	支持
任务导入导出	不支持	支持
工作流实例图	不支持	支持
批量任务操作	不支持	支持
可视化MapReduce（并行计算）	最大300个子任务，无搜索能力	最大1000个子任务，可自定义标签，有搜索能力
历史记录	最近10条任务	最近100条任务
报警	钉钉、邮件	钉钉、邮件、短信、电话、企业微信、飞书
单应用任务数	1000	10万（扩容请提交工单）联系技术支持
OpenAPI	仅支持创建、更新、删除任务的API	所有API都支持
计费项	任务托管CU	任务托管CU和任务调度CU

计费项

SchedulerX的计费项包括任务托管CU和任务调度CU两部分。

任务托管CU

按照任务托管CU总量计算费用，单个SchedulerX应用的计费公式如下：

```
单个SchedulerX应用的任务托管（CU）=启用的单机任务数+启用的非单机任务数×该应用接入的Worker数
```

任务托管总量为您所有应用的任务托管总量之和，且和调度任务每天的调度次数无关。

例如，您有两个SchedulerX应用A和B。

- 应用A接入了10个客户端Worker，启用了8个单机任务和2个非单机任务。
- 应用B接入了5个客户端Worker，启用了2个单机任务和3个非单机任务。

则您每天的任务调度总量为：

```
任务托管总量（CU）=（8+2×10）+（2+3×5）= 45
```

> 注意　启用的任务数只会算当天启用的最大任务数，例如当天创建了1000个任务，删除了1000个任务，同一时间最多启用100个任务，只会算100个任务。

任务调度CU

单次任务调度CU等于任务每次调度的Worker数，例如：

- 单机任务，单次调度CU等于1。
- 非单机任务，单次调度CU等于单次调度的Worker数量。
- 秒级别任务，一分钟最多算一次调度。

任务调度CU总和等于当天所有调度CU总和。

计费方式

基础版（按量付费）

基础版只按照任务托管CU计费，计费方式如下：

计费阶梯	任务托管总量（CU）	单价（元/CU/天）
第一阶梯	0~5	免费
第二阶梯	6~10000	0.1
第三阶梯	10001~50000	0.05
第四阶梯	N（N > 50000）	0.02

专业版（按量付费）

专业版按照任务托管CU和任务调度CU的总和计费。

- 任务托管CU付费方式如下：

计费阶梯	单价（元/CU/天）	单价（元/CU/天）
第一阶梯	0~5	免费
第二阶梯	6~10000	0.1
第三阶梯	10001~50000	0.05
第四阶梯	N（N > 50000）	0.02

- 任务调度CU付费方式如下：

计费阶梯	任务调度CU每天累计数	单价（元/CU）
第一阶梯	0~30	免费
第二阶梯	30~1万	0.008
第三阶梯	1万~5万	0.006
第四阶梯	5万~10万	0.003
第五阶梯	10万~100万	0.001
第六阶梯	100万以上	0.0003

> 注意　秒级别任务一分钟最多算一次调度。

6.3.2. 欠费说明

本文介绍分布式任务调度SchedulerX的欠费停机策略。

欠费停机策略

- 欠费后，控制台中无法新建调度任务。
- 欠费超过2周后，控制台自动禁用所有调度任务。
- 结清账单后，被禁用的调度任务不会自动启用，需要手动启用。
- 如果欠费影响了业务，例如调度任务被禁用，需要提交工单或联系SchedulerX技术支持人员将调度任务手动加入白名单，以免任务被再次禁用。调度任务加入白名单期间，正常计费。

6.4. 快速入门

6.4.1. 准备工作

6.4.1.1. 开通SchedulerX（免费）

在开始使用SchedulerX前，需要先开通。SchedulerX目前在公测期，免费。

背景信息

分布式任务调度目前处于公测期，免费使用。

操作步骤

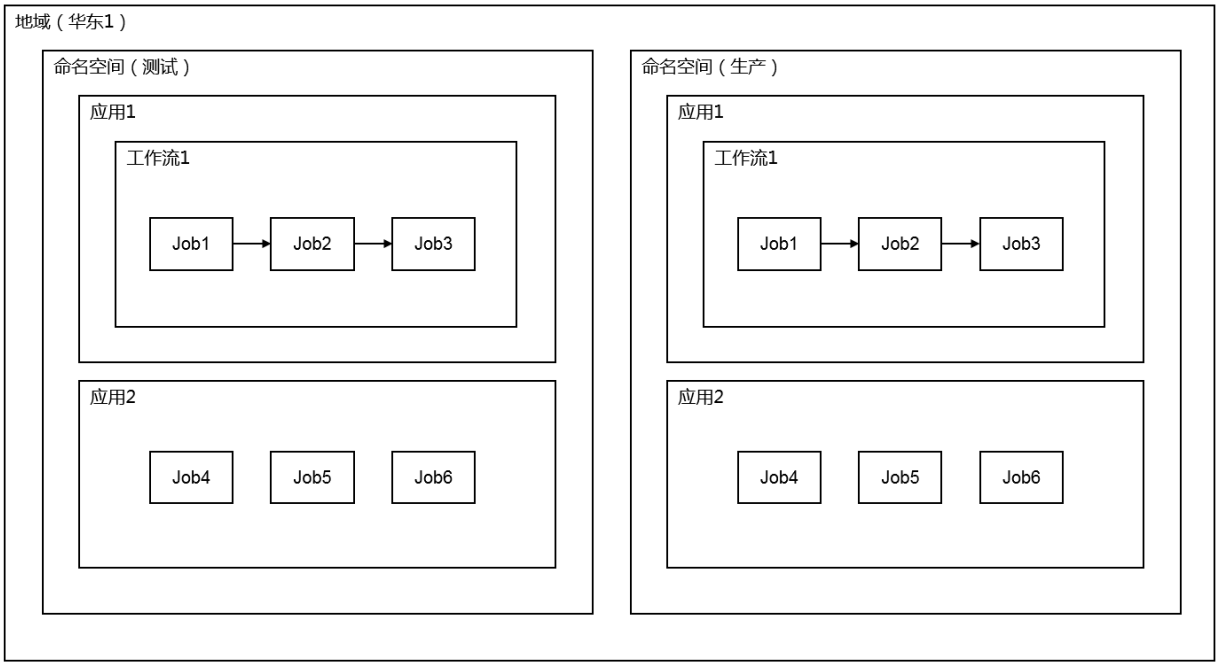
1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 首次使用并登录分布式任务调度平台，在弹出的对话框中单击**开通**。
4. 在**分布式任务管理（按量后付费）**页面单击**立即购买**。

   付费类型为**后付费类型**，**任务计算量**为**按任务计算量付费**，不可修改。
5. 在**确认订单**页面**服务协议**区域单击**我已阅读并同意分布式任务管理（按量后付费）服务协议**，单击**去支付**。

6.4.1.2. 创建资源

在使用SchedulerX前，您需要先创建相关资源，包括命名空间、调度任务分组、调度任务和调度工作流。

背景信息



资源	说明	使用场景
命名空间	在具体地域（Region）中，命名空间用于实现资源和服务的隔离。	当您对资源有较高的安全要求时，需要创建命名空间。
应用	在具体的命名空间下，和应用绑定，关联一组机器。	通过 `GroupId` 绑定应用。
任务	在具体的应用下，任务和一段代码逻辑绑定，用来实现任务调度。	任务是SchedulerX调度的最小单位，用来实现周期性的任务调度。
工作流	在具体应用下，工作流用来实现任务的依赖编排。	工作流是SchedulerX对任务进行依赖编排的封装，支持上下游数据传递。

（可选）创建微服务空间

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**资源管理 > 微服务空间**。
3. 在**微服务空间**页面右上角单击**创建微服务空间**。
4. 在**创建微服务空间**对话框配置微服务空间参数，然后单击**创建**。

参数	描述
微服务空间	请输入您创建的微服务空间的名称。
微服务空间ID	请输入自定义的字符来形成微服务空间的ID，仅允许输入英文字母或数字。
归属地域	当前微服务空间所归属的地域，不可更改。
允许远程调试	当您想对应用进行端云互联时，您在该应用所在的微服务空间的编辑页面手动开启**允许远程调试**。端云调试的相关操作，请参见端云互联简介。
描述	请输入一段文字来描述微服务空间。

创建应用

6.4.2. 客户端快速接入SchedulerX

6.4.2.1. Java应用接入SchedulerX

您可以为您的Java应用快速接入SchedulerX，实现分布式任务调度能力。

前提条件

- （可选）创建命名空间
- 创建应用

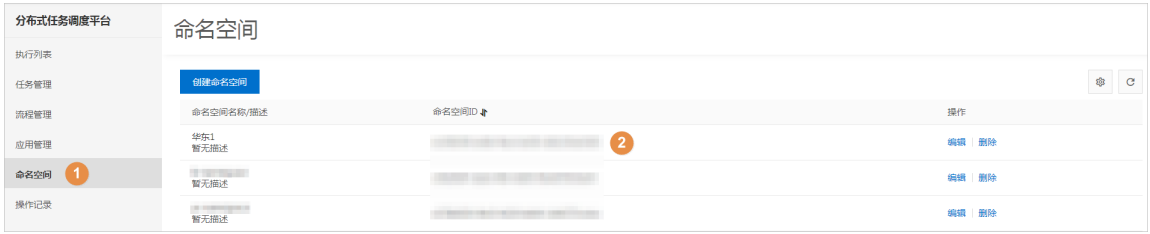
Java应用客户端接入SchedulerX

1. 在应用程序的 `pom.xml` 文件中添加SchedulerxWorker依赖。

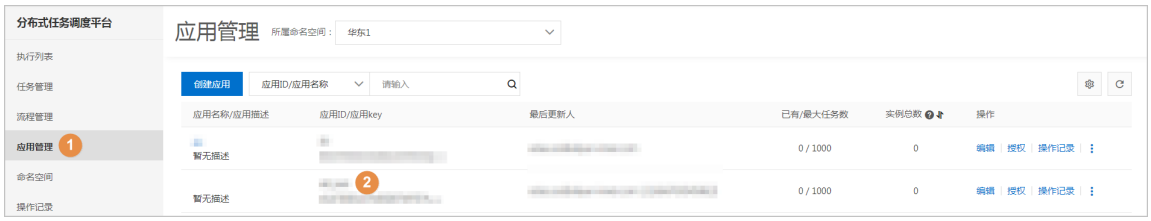
   请参见发布记录， `schedulerx2.version` 使用最新客户端版本。

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-worker</artifactId>
  <version>${schedulerx2.version}</version>
  <!--如果用的是logback，需要把log4j和log4j2排除掉 -->
  <exclusions>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 在main函数中初始化SchedulerxWorker。
   - 初始化SchedulerxWorker时，会用到您部署应用的地域（Region）和对应的Endpoint。详情请参见Endpoint列表。
   - Namespace为命名空间ID，可以在控制台**命名空间**页面获取。



   - GroupId为应用ID，可以在控制台**应用管理**页面获取。



   - AliyunAccessKey和AliyunSecretKey为阿里云账号的AccessKeyID和AccessKeySecret，可以在用户信息管理控制台的**安全信息管理**页面获取。

```
public void initSchedulerxWorker() throws Exception {
    SchedulerxWorker schedulerxWorker = new SchedulerxWorker();
    schedulerxWorker.setEndpoint("xxxx");
    schedulerxWorker.setNamespace("xxxx");
    schedulerxWorker.setGroupId("xxxx");
    //1.2.1及以上版本需要设置应用key
    schedulerxWorker.setAppKey("xxxx");
    //1.2.1以下版本需要设置AK/SK
    //schedulerxWorker.setAliyunAccessKey("xxxx");
    //schedulerxWorker.setAliyunSecretKey("xxxx");
    schedulerxWorker.init();
}
```

> 说明
> - 一个应用如果包含多个业务，或者想把定时任务进行归类，可以建立多个分组，例如应用 `animals` 新建了两个分组 `animals.dogs` 和 `animals.cats` 。此时不用申请两批实例分别接入这两个分组，在应用客户端中将这两个分组配置到 `groupId=` 后面即可，例如 `groupId=animals.dogs,animals.cats` 。
> - 在初始化SchedulerxWorker客户端时，如果还有其它配置需求，可以参考SchedulerxWorker 配置参数说明添加配置。

3. 在应用中创建类 `JobProcessor` ，实现任务调度。

   本文仅介绍如何实现一个最简单的定时打印 `“Hello SchedulerX2.0”` 的 `JobProcessor` 类。

```
package com.aliyun.schedulerx.test.job;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
@Component
public class MyHelloJob extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("hello schedulerx2.0");
        return new ProcessResult(true);
    }
}
```

结果验证

1. 客户端接入完成，将该应用发布到阿里云。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏单击**应用管理**。
4. 在**应用管理**页面查看**实例总数**。
   - 如果**实例总数**为0，则说明应用接入失败。请检查、修改本地应用。
   - 如果**实例总数**不为0，显示接入的实例个数，则说明应用接入成功。在**操作**列单击**查看实例**，即可在**连接实例**对话框中查看实例列表。

后续步骤

应用接入SchedulerX完成后，即可在分布式任务调度平台创建调度任务。详情请参见创建调度任务。

6.4.2.2. Spring应用接入SchedulerX

您可以为您的Spring应用快速接入SchedulerX，实现分布式任务调度能力。

前提条件

- （可选）创建命名空间
- 创建应用

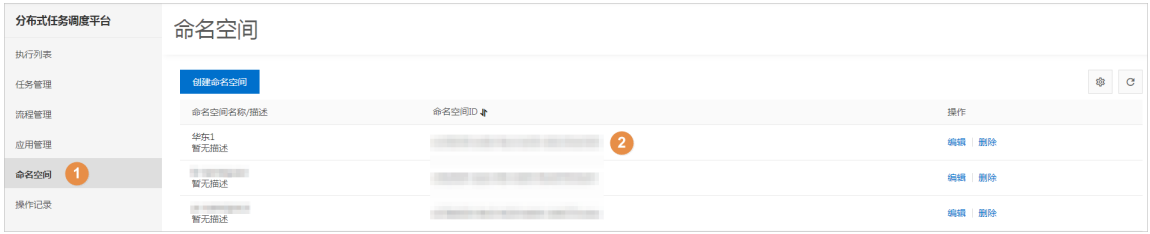
Spring应用客户端接入SchedulerX

1. 在应用程序的 `pom.xml` 文件中添加SchedulerxWorker依赖。

   请参见发布记录， `schedulerx2.version` 使用最新客户端版本。

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-worker</artifactId>
  <version>${schedulerx2.version}</version>
  <!--如果用的是logback，需要把log4j和log4j2排除掉 -->
  <exclusions>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 在xml配置文件中初始化SchedulerxWorker（注入Bean）。
   - 初始化SchedulerxWorker时，会用到您部署应用的地域（Region）和对应的Endpoint。详情请参见Endpoint列表。
   - namespace为命名空间ID，可以在控制台**命名空间**页面获取。



   - groupId为应用ID，appKey为应用key，可以在控制台**应用管理**页面获取。

   - aliyunAccessKey和aliyunSecretKey为阿里云账号的AccessKeyID和AccessKeySecret，可以在用户信息管理控制台的**安全信息管理**页面获取。

```
<bean id="schedulerxWorker" class="com.alibaba.schedulerx.worker.SchedulerxWorker">
    <property name="endpoint">
      <value>${endpoint}</value>
    </property>
    <property name="namespace">
      <value>${namespace}</value>
    </property>
    <property name="groupId">
      <value>${groupId}</value>
    </property>
    <!--1.2.1及以上版本设置appKey -->
    <property name="appKey">
      <value>${appKey}</value>
    </property>
    <!--1.2.1以下版本需要设置AK/SK -->
    <!--
    <property name="aliyunAccessKey">
      <value>${aliyunAccessKey}</value>
    </property>
    <property name="aliyunSecretKey">
      <value>${aliyunSecretKey}</value>
    </property>
    -->
</bean>
```

说明

- 一个应用如果包含多个业务，或者想把定时任务进行归类，可以建立多个分组，例如应用 `animals` 建了两个分组 `animals.dogs` 和 `animals.cats` 。此时不用申请两批实例分别接入这两个分组，在应用客户端中将这两个分组配置到 `groupId=` 后面即可，例如 `groupId=animals.dogs,animals.cats` 。
- 在初始化SchedulerxWorker客户端时，如果还有其它配置需求，可以参考SchedulerxWorker 配置参数说明添加配置。

3. 在应用中创建类 `JobProcessor` ，实现任务调度。

   本文仅介绍如何实现一个最简单的定时打印 `“Hello SchedulerX2.0”` 的 `JobProcessor` 类。

```
package com.aliyun.schedulerx.test.job;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
@Component
public class MyHelloJob extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("hello schedulerx2.0");
        return new ProcessResult(true);
    }
}
```

结果验证

1. 客户端接入完成，将该应用发布到阿里云。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏单击**应用管理**。

4. 在**应用管理**页面查看**实例总数**。
   - 如果**实例总数**为0，则说明应用接入失败。请检查、修改本地应用。
   - 如果**实例总数**不为0，显示接入的实例个数，则说明应用接入成功。在**操作**列单击**查看实例**，即可在**连接实例**对话框中查看实例列表。

后续步骤

应用接入SchedulerX完成后，即可在分布式任务调度平台创建调度任务。详情请参见创建调度任务。

6.4.2.3. Spring Boot应用接入SchedulerX

您可以为您的Spring Boot应用快速接入SchedulerX，实现分布式任务调度能力。

前提条件

- （可选）创建命名空间
- 创建应用

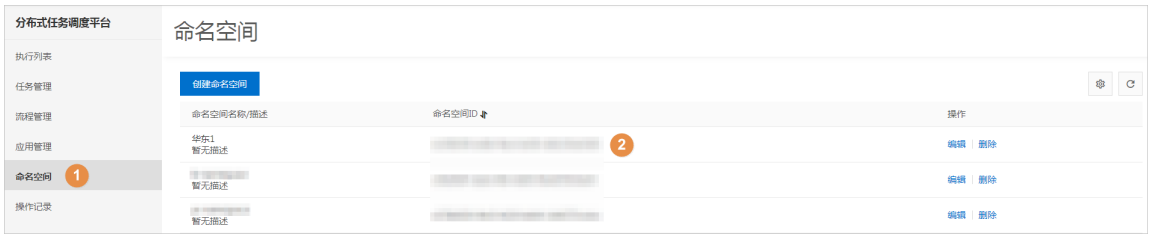
Spring Boot应用客户端接入SchedulerX

1. 在应用程序的 `pom.xml` 文件中添加SchedulerxWorker依赖。

   请参见版本发布记录， `schedulerx2.version` 使用最新客户端版本。

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-spring-boot-starter</artifactId>
  <version>${schedulerx2.version}</version>
  <!--如果用的是logback，需要把log4j和log4j2排除掉 -->
  <exclusions>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 在 `application.properties` 文件中设置相关参数，初始化SchedulerxWorker。
   - 初始化SchedulerxWorker时，会用到您部署应用的地域（Region）和对应的Endpoint。详情请参见Endpoint列表。
   - namespace为命名空间ID，可以在控制台**命名空间**页面获取。



   - groupId为应用ID，appKey为应用key，可以在控制台**应用管理**页面获取。

- aliyunAccessKey和aliyunSecretKey为阿里云账号的AccessKeyID和AccessKeySecret，可以在用户信息管理控制台的**安全信息管理**页面获取。

```
spring.schedulerx2.endpoint=${endpoint}
spring.schedulerx2.namespace=${namespace}
spring.schedulerx2.groupId=${groupId}
# 1.2.1及以上版本设置appKey
spring.schedulerx2.appKey=${appKey}
# 1.2.1以下版本设置AK/SK
#spring.schedulerx2.aliyunAccessKey=${aliyunAccessKey}
#spring.schedulerx2.aliyunSecretKey=${aliyunSecretKey}
```

说明 一个应用如果包含多个业务，或者想把定时任务进行归类，可以建立多个分组，例如应用 `animals` 新建了两个分组 `animals.dogs` 和 `animals.cats` 。此时不用申请两批实例分别接入这两个分组，在应用客户端中将这两个分组配置到 `groupId=` 后面即可，例如 `groupId=animals.dogs,animals.cats` 。

如果还有其它需求，请参考下表中的参数进行配置。

key	描述	设置值	起始版本
spring.schedulerx2.enabled	是否启用SchedulerX 2.0的starter，默认开启不需要设置。	true/false，默认true。	0.1.7
spring.schedulerx2.endpoint	设置Region所在的Endpoint，详情请参见Endpoint列表。	无	0.1.7
spring.schedulerx2.namespace	设置命名空间的UID，可以在控制台**命名空间**页面获取。	无	0.1.7
spring.schedulerx2.groupId	应用ID，可以在控制台**应用管理**页面获取。	无	0.1.7
spring.schedulerx2.appKey	应用key，可以在控制台**应用管理**页面获取。	无	1.2.1
spring.schedulerx2.host	如果有多个IP（例如VPN或者多网卡），可以设置真实的IP。	无	0.1.7
spring.schedulerx2.port	用户自定义客户端监听端口。如果不设置，则随机选择一个可用端口。	无	0.1.7
spring.schedulerx2.blockAppStart	SchedulerX初始化失败是否block应用启动，默认为True。	true/false，默认true。	1.1.0
spring.schedulerx2.shareContainerPool	客户端所有任务执行是否共享线程池，默认为False。	无	1.2.1.2

key	描述	设置值	起始版本
spring.schedulerx2.sharePoolSize	如果开启共享线程池，可以自定义线程池大小，默认为64。	无	1.2.1.2
spring.schedulerx2.label	不同客户端可以设置标签，任务管理可以指定标签执行。应用于灰度、压测等场景。	无	1.2.2.2
spring.schedulerx2.enableCgroupMetrics	是否使用cgroup统计客户端实例的指标。容器（k8s）环境需要自己手动开启。	true/false，默认false。	1.2.2.2
spring.schedulerx2.cgroupPathPrefix	容器内cgroup的路径。	默认是/sys/fs/cgroup/cpu/，如果存在该路径则不需要设置。	1.2.2.2
spring.schedulerx2.enableHeartbeatLog	是否打印心跳日志，${user.home}/logs/schedulerx/heartbeat.log	true/false，默认true	1.2.4
spring.schedulerx2.mapMasterStatusCheckInterval	设置Map模型检测所有子任务结束的频率，单位毫秒。如果是秒级别任务想要加快调度频率，可以设置。	3000	1.2.5.2
spring.schedulerx2.enableSecondDealyCycleIntervalMs	设置second_delay延迟的单位为毫秒。如果把这个值设置为true，控制台设置的秒级别延迟将会变成毫秒，可以加快调度频率。	true/false，默认false	1.2.5.2

3. 在应用中创建类 `JobProcessor` ，实现任务调度。

   本文仅介绍如何实现一个最简单的定时打印 `“Hello SchedulerX2.0”` 的 `JobProcessor` 类。

```
package com.aliyun.schedulerx.test.job;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
@Component
public class MyHelloJob extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("hello schedulerx2.0");
        return new ProcessResult(true);
    }
}
```

结果验证

1. 客户端接入完成，将该应用发布到阿里云。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏单击**应用管理**。

4. 在**应用管理**页面查看**实例总数**。
   - 如果**实例总数**为0，则说明应用接入失败。请检查、修改本地应用。
   - 如果**实例总数**不为0，显示接入的实例个数，则说明应用接入成功。在**操作**列单击**查看实例**，即可在**连接实例**对话框中查看实例列表。

后续步骤

应用接入SchedulerX完成后，即可在分布式任务调度平台创建调度任务。详情请参见创建调度任务。

6.4.2.4. Agent接入（调度任务）

如果您无需为应用接入调度任务，仅想创建一个独立的脚本调度任务，也可以使用SchedulerX提供的Agent快速创建脚本任务。

前提条件

- （可选）创建命名空间
- 创建应用

背景信息

脚本任务目前支持Shell、Python和Go三种语言。

运行环境要求为JRE 1.8及以上版本。

操作步骤

1. 下载schedulerxAgent-1.4.1。
2. 解压下载的压缩包。
3. 进入*schedulerxAgent/conf*目录，编辑*agent.properties*文件，添加endpoint、namespace（命名空间ID）、groupId（应用ID）和appKey（应用Key）。

   ```
   endpoint=
   namespace=
   groupId=
   appKey=
   ```

   地域（Region）和Endpoint的关系请参见Endpoint列表。
4. 进入*schedulerxAgent/bin*目录，执行*start-1g.sh*命令启动SchedulerX。

   > **说明** *start-1g.sh*仅为示例，您需要根据任务负载及机器配置情况执行对应的命令，如*start-2g.sh*、*start-4g.sh*或*start-8g.sh*。

   如果您想停止任务调度，可执行*stop.sh*命令。

6.4.2.5. 在本地接入公网环境

本文介绍如何将本地接入云上公网环境来进行本地开发、调试和验证。

前提条件

- 开通SchedulerX
- 创建应用

> **说明** 在创建应用时，请选择**公网**地域。

操作步骤

1. 在应用*pom.xml*文件中添加SchedulerxWorker依赖。

针对不同应用，在初始化SchedulerxWorker时会有所不同。

- 普通Java或Spring应用

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-worker</artifactId>
  <version>${schedulerx2.version}</version>
  <!--如果用的是logback，需要把log4j和log4j2排除掉 -->
  <exclusions>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

- Spring Boot应用

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-spring-boot-starter</artifactId>
  <version>${schedulerx2.version}</version>
  <!--如果用的是logback，需要把log4j和log4j2排除掉 -->
  <exclusions>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 初始化SchedulerxWorker。

   针对不同应用，在初始化SchedulerxWorker的时候会有所不同。

   - 普通Java应用
     在main函数中初始化SchedulerxWorker。
     初始化SchedulerxWorker时，会用到您部署应用的**公网**地域（Region）和对应的Endpoint（**acm.aliyun.com**）。

     a. 在控制台顶部菜单栏将地域切换为**公网**。

b. 在左侧导航栏单击**命名空间**，在**命名空间**页面查看**命名空间ID**（Namespace）。

c. 在左侧导航栏单击**应用管理**，在**应用管理**页面查看**应用ID**（groupId）和**应用key**（appKey）。

d. 将光标移动到控制台右上角的用户头像上，在弹出的列表中单击**AccessKey管理**，在**AccessKey管理**页面获取AccessKeyID（AliyunAccessKey）和AccessKeySecret（AliyunSecretKey）。

```
public void initSchedulerxWorker() throws Exception {
    SchedulerxWorker schedulerxWorker = new SchedulerxWorker();
    schedulerxWorker.setEndpoint("xxxx");
    schedulerxWorker.setNamespace("xxxx");
    schedulerxWorker.setGroupId("xxxx");
    //1.2.1及以上版本需要设置应用key
    schedulerxWorker.setAppKey("xxxx");
    //1.2.1以下版本需要设置AK/SK
    //schedulerxWorker.setAliyunAccessKey("xxxx");
    //schedulerxWorker.setAliyunSecretKey("xxxx");
    schedulerxWorker.init();
}
```

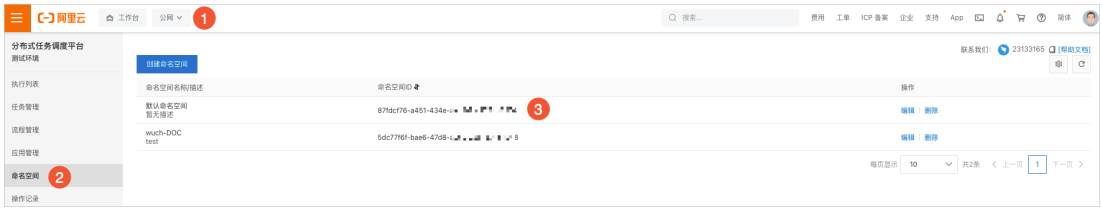
说明

- 一个应用如果包含多个业务，或者想把定时任务进行归类，可以建立多个分组，例如应用 `animals` 新建了两个分组 `animals.dogs` 和 `animals.cats` 。此时不用申请两批实例分别接入这两个分组，在应用客户端中将这两个分组配置到 `groupId=` 后面即可，例如 `groupId=animals.dogs,animals.cats` 。
- 在初始化SchedulerxWorker客户端时，如果还有其他配置需求，可以添加配置。更多信息，请参见SchedulerxWorker 配置参数说明添加配置。

- Spring应用
  在xml配置文件中注入SchedulerxWorker Bean。

a. 在控制台顶部菜单栏将地域切换为**公网**。

b. 在左侧导航栏单击**命名空间**，在**命名空间**页面查看**命名空间ID**（Namespace）。



c. 在左侧导航栏单击**应用管理**，在**应用管理**页面查看**应用ID**（groupId）和**应用key**（appKey）。

d. 将光标移动到控制台右上角的用户头像上，在弹出的列表中单击**AccessKey管理**，在**AccessKey管理**页面获取AccessKeyID（AliyunAccessKey）和AccessKeySecret（AliyunSecretKey）。

```
<bean id="schedulerxWorker" class="com.alibaba.schedulerx.worker.SchedulerxWorker">
    <property name="endpoint">
      <value>${endpoint}</value>
    </property>
    <property name="namespace">
      <value>${namespace}</value>
    </property>
    <property name="groupId">
      <value>${groupId}</value>
    </property>
    <!--1.2.1及以上版本设置appKey -->
    <property name="appKey">
      <value>${appKey}</value>
    </property>
    <!--1.2.1以下版本需要设置AK/SK -->
    <!--
    <property name="aliyunAccessKey">
      <value>${aliyunAccessKey}</value>
    </property>
    <property name="aliyunSecretKey">
      <value>${aliyunSecretKey}</value>
    </property>
    -->
</bean>
```

- Spring Boot应用
  在*application.properties*文件中添加如下配置：

a. 在控制台顶部菜单栏将地域切换为**公网**。

b. 在左侧导航栏单击**命名空间**，在**命名空间**页面查看**命名空间ID**（Namespace）。

c. 在左侧导航栏单击**应用管理**，在**应用管理**页面查看**应用ID**（groupId）和**应用key**（appKey）。

d. 将光标移动到控制台右上角的用户头像上，在弹出的列表中单击**AccessKey管理**，在**AccessKey管理**页面获取AccessKeyID（AliyunAccessKey）和AccessKeySecret（AliyunSecretKey）。

```
spring.schedulerx2.endpoint=${endpoint}
spring.schedulerx2.namespace=${namespace}
spring.schedulerx2.groupId=${groupId}
# 1.2.1及以上版本设置appKey
spring.schedulerx2.appKey=${appKey}
# 1.2.1以下版本设置AK/SK
#spring.schedulerx2.aliyunAccessKey=${aliyunAccessKey}
#spring.schedulerx2.aliyunSecretKey=${aliyunSecretKey}
```

> **说明** 一个应用如果包含多个业务，或者想把定时任务进行归类，可以建立多个分组，例如应用 `animals` 新建了两个分组 `animals.dogs` 和 `animals.cats` 。此时不用申请两批实例分别接入这两个分组，在应用客户端中将这两个分组配置到 `groupId=` 后面即可，例如 `groupId=animals.dogs,animals.cats` 。

3. 在应用中创建类 `JobProcessor` ，实现任务调度。

   本文仅介绍如何实现一个最简单的定时打印 `“Hello SchedulerX2.0”` 的 `JobProcessor` 类。

```
package com.aliyun.schedulerx.test.job;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
@Component
public class MyHelloJob extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("hello schedulerx2.0");
        return new ProcessResult(true);
    }
}
```

4. 运行本地应用。

结果验证

1. 客户端接入完成，将该应用发布到阿里云。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏单击**应用管理**。
4. 在**应用管理**页面查看**实例总数**。
   - 如果**实例总数**为0，则说明应用接入失败。请检查、修改本地应用。
   - 如果**实例总数**不为0，显示接入的实例个数，则说明应用接入成功。在**操作**列单击**查看实例**，即可在**连接实例**对话框中查看实例列表。

6.4.2.6. 容器服务Kubernetes版接入SchedulerX

前提条件

- 创建Kubernetes专有版集群
- 开通SchedulerX
- 了解Kubernetes基本概念和操作，例如kubeconfig

背景信息

使用SchedulerX需要理解以下3个概念：

- 分组
  - 客户端的组织单位
  - 任务的组织单位
- 任务：调度单位，需要创建任务，配置所属分组。
- 客户端：任务执行节点，需要添加SchedulerX客户端，实现对应Java任务处理接口，配置所属分组启动名为SchedulerxWorker的Agent。

三者的关系为：任务只能调度到对应分组的客户端。例如创建了分组group-sample，在该分组下创建任务job-sample，同时配置所属分组为group-sample来启动客户端agent1、agent2和agent3，则调度任务job-sample就会被调度到客户端agent1、agent2和agent3上面运行。

关于SchedulerX的更多信息，请参见分布式任务调度 SchedulerX。

在容器服务Kubernetes版中安装SchedulerX组件

1. 登录容器服务控制台。
2. 在左侧导航栏选择**市场 > 应用市场**。
3. 在**应用市场**页面搜索并单击**ack-schedulerx**，仔细阅读**说明**。
4. 在**ack-schedulerx**页面右侧单击**一键部署**。

   配置参数后，单击**下一步**，设置参数controller.aliyun_acccessKey_secret_name。

   > 说明　命名空间为schedulerx-system，不可修改。

   安装SchedulerX组件大约需要2分钟，请耐心等待。
   创建成功后，会自动跳转到**Helm 发布列表 - ack-schedulerx**页面，检查安装结果。

创建分组

1. 创建*xgroup.yaml*，设置相关参数。

   *xgroup.yaml*示例如下：

   ```
   apiVersion: schedulerx.alibabacloud.com/v1alpha1
   kind: XGroup
   metadata:
     name: xgroup-sample
   spec:
     appName: ackApp
   ```

   *xgroup.yaml*配置参数说明：

   - GVK（Group、Version 和 Kind）信息
     - apiVersion：格式为 `<group>/<version>` ，例如 `schedulerx.alibabacloud.com/v1alpha1` 。
     - kind: XGroup
   - spec信息

参数名	类型	默认值	是否必填	说明
appName	string	无	必填	应用名，用户自定义，用于后续管理。

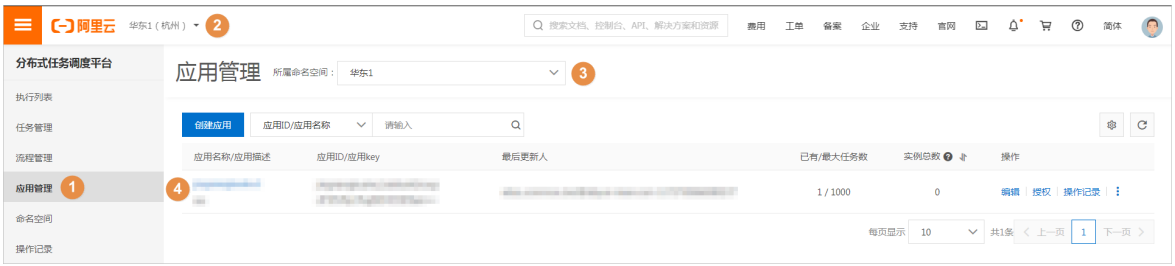
2. 执行 `kubectl apply -f xgroup.yaml` 命令，创建任务分组。

> **说明**
>
> - 分组创建后，不允许更新。如果需要更新，请删除分组后重新创建。
> - 分组创建后，如果新建了任务或添加了客户端，即该分组下包含任务或客户端，则无法删除分组，需要先移除任务和客户端。

3. 执行 `kctl get xgroup xgroup-sample -o yaml` 命令，查看xgroup资源。
   打印结果如下：

```
apiVersion: schedulerx.alibabacloud.com/v1alpha1
kind: XGroup
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"schedulerx.alibabacloud.com/v1alpha1","kind":"XGroup","metadata":{"annotati
ons":{},"name":"xgroup-sample","namespace":"default"},"spec":{"appName":"ackApp"}}
  creationTimestamp: "2019-09-19T04:21:12Z"
  finalizers:
  - GroupCleanup
  generation: 1
  name: xgroup-sample
  namespace: default
  resourceVersion: "143176160"
  selfLink: /apis/schedulerx.alibabacloud.com/v1alpha1/namespaces/default/xgroups/xgroup-sample
  uid: e9a1xxxx-xxxx-11e9-xxxx-be9f1a43xxxx
spec:
  appName: ackApp
status:
  appGroupId: 283
  conditions:
  - lastTransitionTime: "2019-09-19T04:21:12Z"
    lastUpdateTime: "2019-09-19T04:21:12Z"
    reason: CreateGroupSuccess
    status: "True"
    type: Ready
```

status的conditions下status为True，且type为Ready，则表示分组创建成功。
也可以登录分布式任务调度平台，查看分组创建情况。



创建任务

1. 创建*xcronjob.yaml*，设置相关参数。
   *xcronjob.yaml*示例如下：

```
apiVersion: schedulerx.alibabacloud.com/v1alpha1
kind: XCronJob
metadata:
  name: xcronjob-sample
spec:
  group: xgroup-sample
  jobType: java
  jobProcessor: processor.SimpleJobProcessor
  executeMode: standalone
  timeExpression: 0 0 2 * * ?
```

*xcronjob.yam*配置参数说明：

- GVK（Group、Version 和 Kind）信息
  - apiVersion：格式为 `<group>/<version>`，例如 `schedulerx.alibabacloud.com/v1alpha1`。
  - kind: XCronJob
- spec信息

参数名	类型	默认值	是否必填	说明
group	string	无	是	该任务所属分组名。
jobType	string	java	否	任务类型，指实现任务的编程语言，当前支持Java、Python、Shell和Go。详情请参见多种调度任务类型。
jobProcessor	string	无	否（有条件）	任务实现全限定类名，如果 `jobType == java`，该字段必填。
content	string	无	否（有条件）	任务实现代码，如果 `jobType == java`，该字段必填。
executeMode	string	standalone	否	任务执行模式，当前支持单击运行、广播运行、并行计算、内存网格、网格计算和分片运行。详情请参见分布式计算。
description	string	无	否	任务描述
timeType	int	1	否	任务调度表达式类型，当前支持cron(1)、fix_rate(3)、second_delay(4)。详情请参见Cron、Fixed rate和Second delay。

参数名	类型	默认值	是否必填	说明
timeExpression	string	无	是	任务调度表达式，例如： ■ cron：`0 0 2 * * ?` 要确保频率大于分钟级。 ■ fixed_rate：`30(>0)`，单位是s，每30s运行一次。 ■ second_delay：`2(1-60)`，单位是s，上次运行结束后延迟2s再运行一次。
parameters	string	无	否	任务参数，可以在任务运行时从上下文获取。
maxConcurrency	int	1	否	最大同时运行任务实例数，默认为1。超过该并发度的调度实例会被忽略。
retryMaxAttempts	int	0	否	失败重试次数，默认为0，不重置。
retryInterval	int	30	否	失败重试间隔，单位s，默认为30s。

通过示例可以看到指定的group是刚刚创建的xgroup-sample，默认使用Cron调度表达式，Java任务类型，处理的接口类名为processor.SimpleJobProcessor。

2. 执行 `kubectl apply -f xcronjob.yaml` 命令，创建任务。
3. 执行 `kctl get xcronjob xcronjob-sample -o yaml` 命令，查看xcronjob资源。
   打印结果如下：

```
apiVersion: schedulerx.alibabacloud.com/v1alpha1
kind: XCronJob
metadata:
  creationTimestamp: "2019-09-19T06:33:13Z"
  finalizers:
  - JobCleanup
  generation: 1
  name: xcronjob-sample
  namespace: default
  ownerReferences:
  - apiVersion: schedulerx.alibabacloud.com/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: XGroup
    name: xgroup-sample
    uid: e9a1xxxx-xxxx-11e9-xxxx-be9f1a43xxxx
  resourceVersion: "143570391"
  selfLink: /apis/schedulerx.alibabacloud.com/v1alpha1/namespaces/default/xcronjobs/xcronjob-sam
ple
  uid: 5b5exxxx-daa7-xxxx-a76d-4af3xxxx44xx
spec:
  executeMode: standalone
  group: xgroup-sample
  jobProcessor: processor.SimpleJobProcessor
  jobType: java
  timeExpression: 0 0 2 * * ?
status:
  conditions:
  - lastTransitionTime: "2019-09-19T06:33:13Z"
    lastUpdateTime: "2019-09-19T06:33:14Z"
    reason: JobUpdateSuccess
    status: "True"
    type: Ready
  jobId: 1304
```

status的conditions下status为True，且type为Ready，则表示任务创建成功。
也可以登录分布式任务调度平台，查看任务创建情况。

创建客户端

1. 客户端接入SchedulerX。

   根据应用类型不同，客户端的接入方式也不同。

   - Java应用：客户端接入详情请参见Java应用接入SchedulerX。
   - Spring应用：客户端接入详情请参见Spring应用接入SchedulerX。
   - Spring Boot应用：客户端接入详情请参见Spring Boot应用接入SchedulerX。

2. 将客户端打包，并通过Dockerfile创建镜像，并上传到镜像仓库。

   Dockfile示例如下：

```
FROM openjdk:8-jdk-alpine
COPY ./target/schedulerx-k8s-demo-1.0-SNAPSHOT-spring-boot.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

SchedulerX提供了已经上传到镜像仓库的镜像Demo（`registry.cn-shanghai.aliyuncs.com/schedulerx/demo:latest`），方便您体验、试用。

3. 创建*xagentpool.yaml*，设置相关参数。

   *xagentpool.yaml*示例如下：

```
apiVersion: schedulerx.alibabacloud.com/v1alpha1
kind: XAgentPool
metadata:
  name: xagentpool-sample
spec:
  group: xgroup-sample
  replicas: 2
  template:
    containers:
      - name: standalone
        image: registry.cn-shanghai.aliyuncs.com/schedulerx/demo:latest
```

   *xagentpool.yaml*配置参数说明：

   - GVK（Group、Version 和 Kind）信息
     - apiVersion：格式为`<group>/<version>`，例如`schedulerx.alibabacloud.com/v1alpha1`。
     - kind: XGroup
   - spec信息

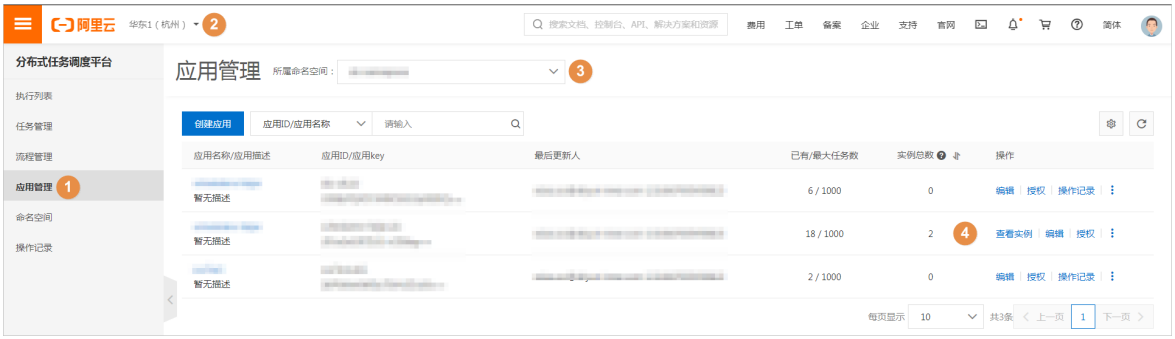
参数名	类型	默认值	是否必填	说明
group	string	无	是	该任务所属分组名。
replicas	int	无	是	执行器个数。
workloadType	string	Deployment	否	执行器工作负载，默认是Deployment，可选值还有DaemonSet。
template	PodSpec	无	是	任务执行器Pod模板。

   通过示例可以看到指定的group是刚刚创建的xgroup-sample，运行两个执行器，执行器镜像为image，该image即客户端镜像。

4. 执行`kubectl apply -f xagentpool.yaml`命令，创建客户端。
5. 执行`kctl get xagentpool xagentpool-sample -o yaml`命令，查看xagentpool资源。
   打印结果如下：

```
apiVersion: schedulerx.alibabacloud.com/v1alpha1
kind: XAgentPool
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"schedulerx.alibabacloud.com/v1alpha1","kind":"XAgentPool","metadata":{"anno
tations":{},"name":"xagentpool-sample","namespace":"default"},"spec":{"group":"xgroup-sample","r
eplicas":2,"template":{"containers":[{"image":"registry.cn-shanghai.aliyuncs.com/schedulerx/demo
:latest","name":"standalone"}]}}}
  creationTimestamp: "2019-09-25T10:11:39Z"
  generation: 1
  name: xagentpool-sample
  namespace: default
  ownerReferences:
  - apiVersion: schedulerx.alibabacloud.com/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: XGroup
    name: xgroup-sample
    uid: c920xxxx-df7c-xxxx-a76d-xxxx350bxxxx
  resourceVersion: "170986882"
  selfLink: /apis/schedulerx.alibabacloud.com/v1alpha1/namespaces/default/xagentpools/xagentpool
-sample
  uid: dd83xxxx-df7c-xxxx-a156-xxxx1a43xxxx
spec:
  group: xgroup-sample
  replicas: 2
  template:
    containers:
    - image: registry.cn-shanghai.aliyuncs.com/schedulerx/demo:latest
      name: standalone
      resources: {}
status:
  conditions:
  - lastTransitionTime: "2019-09-25T10:11:40Z"
    lastUpdateTime: "2019-09-25T10:11:40Z"
    reason: update deployment sucess
    status: "True"
    type: Ready
```

status的conditions下status为True，且type为Ready，则表示任务创建成功。
实际上，每个agentPool的创建都会在相同命名空间下创建名为 `[agentPoolName]-deployment` 的Deployment或者 `[agentPoolName]-daemonset` 的DeamonSet，可以自行查看
也可以登录分布式任务调度平台，查看客户端创建情况。



单击**查看实例**，可以查看客户端实例的详细信息。

删除SchedulerX组件

当不再需要SchedulerX组件时，可以删除组件。

> 说明
>
> - 在删除SchedulerX组件之前，请确保集群内的所有XGroup、XCronJob和XAgentPool类型资源都已经删除完毕，否则无法删除CRDs，并且会导致下次安装出现异常。
> - XGroup删除之后，对应的SchedulerX应用分组不会自动删除，需要登录分布式任务调度平台，手动删除。

1. 登录容器服务控制台。
2. 在左侧导航栏选择**应用 > 发布**。
3. 在**发布**页面**ack-schedulerx**的**操作**列单击**删除**。
4. 在**删除应用**对话框中单击**确定**。

6.4.2.7. 在Kubernetes集群中部署SchedulerX

本文介绍如何在Kubernetes环境中部署SchedulerX，通过SchedulerX可以定时调度您的程序、多语言脚本和HTTP接口，也可以调度原生的K8s Job或者Pod。

前提条件

- 已存在Kubernetes集群（阿里云容器服务ACK集群或自建Kubernetes集群）。
- 开通SchedulerX。
- 在分布式任务调度平台创建一个应用类型为K8s的应用。

背景信息

使用SchedulerX调度K8s Job有如下优势：

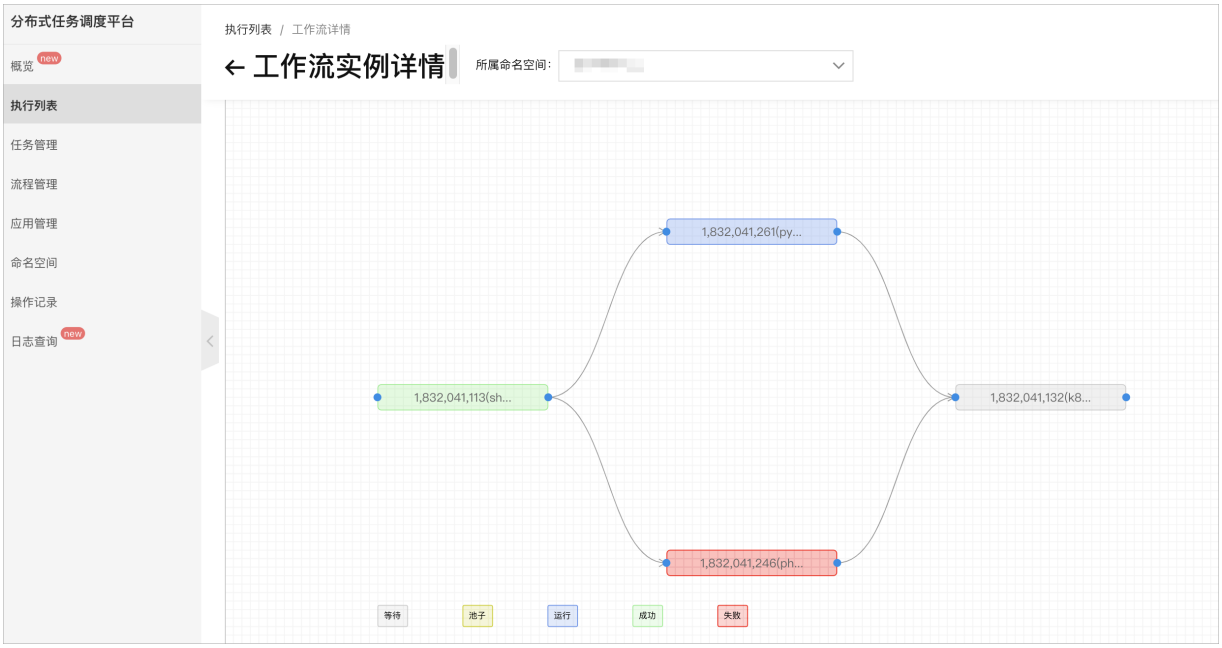
可在线编辑的脚本Pod

K8s Job常用场景是用来做数据处理和运维，一般以脚本实现居多。原生的使用方式需要把脚本打包到镜像里，在YAML文件中配置脚本命令。如果要修改脚本，就需要重新构建镜像和发布，操作比较复杂。

而使用SchedulerX则无需构建镜像和写YAML脚本，只需要在SchedulerX控制台直接编辑脚本（Shell、Python、PHP和Node.js），即可自动以Pod方式运行脚本。如果要修改脚本，只需要在SchedulerX控制台重新编辑脚本，下次调度自动生效，从而提高了K8s Job的开发效率。同时，使用SchedulerX的K8s任务屏蔽了容器相关的细节，为不熟悉容器服务的用户提供便利。

可视化任务编排

SchedulerX支持通过可视化界面拖拽进行K8s任务的编排，相比于当前主流的通过代码进行工作流编排的解决方案来说，使用更便捷。而且在运行时，可视化的工作流图可以帮助您快速排查任务卡在哪个环节。



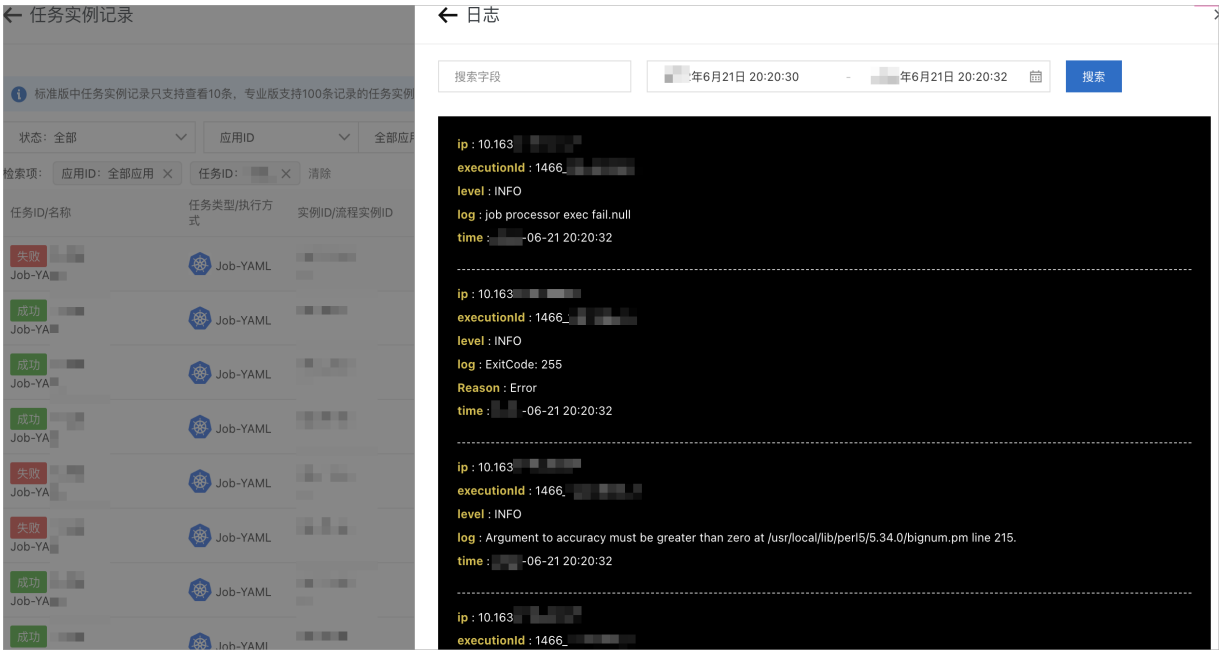
报警监控

使用SchedulerX来调度您的Pod或者Job，可以复用SchedulerX的监控报警功能。

- 支持的报警通道：短信、电话、邮件、Webhook（钉钉/企业微信/飞书）。
- 支持的报警策略：失败报警、执行超时报警。

日志服务

使用SchedulerX来调度您的Pod或者Job，不需要额外开通日志服务，可以自动采集Pod运行的日志，如果Pod运行失败，可以直接在SchedulerX控制台看到Pod执行失败的原因。



监控大盘

您可以通过SchedulerX自带的任务监控大盘实时观察您的任务状态。

离在线混布

SchedulerX支持的Java和K8s任务类型，可以做到离在线定时任务混布调度。一个业务应用通常有很多定时任务，如果调度频率比较高，可以直接和业务应用在一个进程中，但进程内调用会消耗在线应用自身的cpu和内存，无法和在线业务做隔离。所以当一个定时任务非常耗资源，调度频率又不高（比如每小时/每天运行一次）时，可以通过新增一个Pod去运行，这样就和原来的在线应用不在一个进程中了。

步骤一：配置ServiceAccount

SchedulerX K8s任务依赖于ServiceAccount进行验证与授权，而且默认情况下，它使用所在Namespace的SchedulerX ServiceAccount运行K8s任务。

在K8s集群里和对应的Namespace下，运行schedulerx-serviceaccount.yaml（运行一次即可），格式参考如下：

- 如果每个Namespace做隔离，只能调度自己Namespace下的Pod或者Job，只需要运行如下YAML。
  展开查看具体代码

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: schedulerx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: schedulerx-role
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create","delete","get","list","patch","update","watch"]
  - apiGroups: [""]
    resources: ["pods/exec"]
    verbs: ["create","delete","get","list","patch","update","watch"]
  - apiGroups: [""]
    resources: ["pods/log"]
    verbs: ["get","list","watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["create","delete","get","list","patch","update"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["watch"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["create","delete","get","list","patch","update","watch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: schedulerx-binding
subjects:
  - kind: ServiceAccount
    name: schedulerx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: schedulerx-role
```

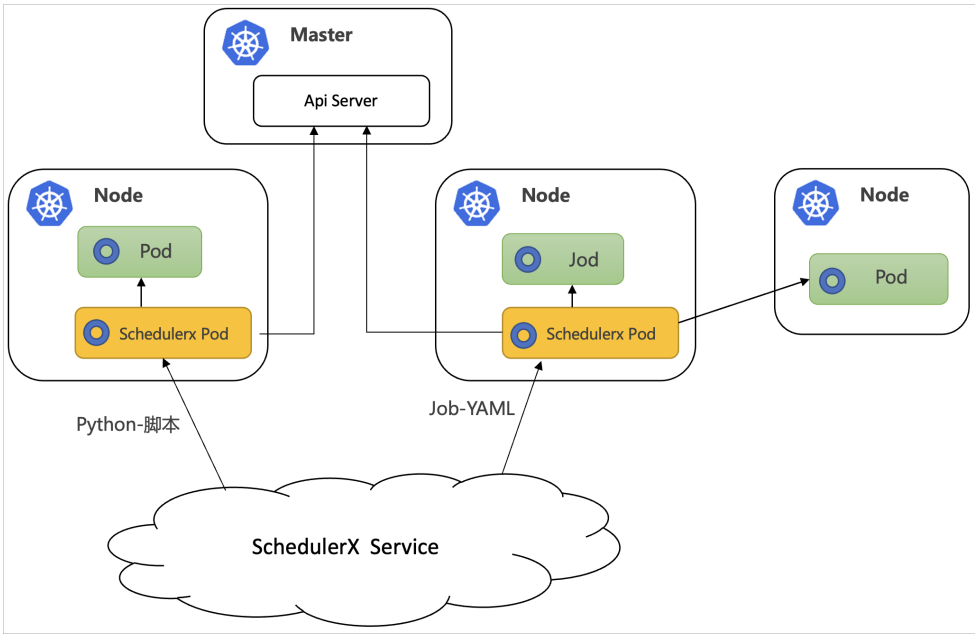
- 如果有跨Namespace调度的需求，RoleBinding需要配置多个Namespace。
  展开查看具体代码

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: schedulerx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: schedulerx-role
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create","delete","get","list","patch","update","watch"]
  - apiGroups: [""]
    resources: ["pods/exec"]
    verbs: ["create","delete","get","list","patch","update","watch"]
  - apiGroups: [""]
    resources: ["pods/log"]
    verbs: ["get","list","watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["create","delete","get","list","patch","update"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["watch"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["create","delete","get","list","patch","update","watch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: schedulerx-binding
subjects:
  - kind: ServiceAccount
    name: schedulerx
    namespace: <NAMESPACE1>
  - kind: ServiceAccount
    name: schedulerx
    namespace: <NAMESPACE2>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: schedulerx-role
```

步骤二：接入SchedulerX

通过Deployment部署

如果您是非Java应用，可以通过Deployment部署一个schedulerx-agent.yaml，这样SchedulerX会以单独的Pod启动，原理图如下：



展开查看schedulerx-agent.yaml配置

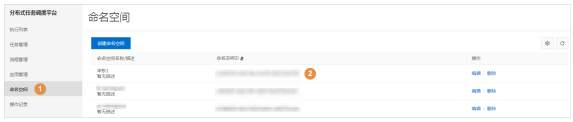
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: schedulerx-agent
  labels:
    app: schedulerx-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app: schedulerx-agent
  template:
    metadata:
      labels:
        app: schedulerx-agent
    spec:
      serviceAccountName: schedulerx
      containers:
      - name: schedulerx-agent
        image: registry.cn-huhehaote.aliyuncs.com/schedulerx/agent:1.6.0-amd64
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 200m
          requests:
            cpu: 200m
        env:
          - name: "SCHEDULERX_ENDPOINT"
            value: "${SCHEDULERX_ENDPOINT}"
          - name: "SCHEDULERX_NAMESPACE"
            value: "${SCHEDULERX_NAMESPACE}"
          - name: "SCHEDULERX_GROUPID"
            value: "${SCHEDULERX_GROUPID}"
          - name: "SCHEDULERX_APPKEY"
            value: "${SCHEDULERX_APPKEY}"
          - name: "SCHEDULERX_STARTER_MODE"
            value: "pod"
        livenessProbe:
          exec:
            command: ["/bin/bash","/root/health.sh"]
          timeoutSeconds: 30
          initialDelaySeconds: 30
```

SchedulerX agent image变量说明：

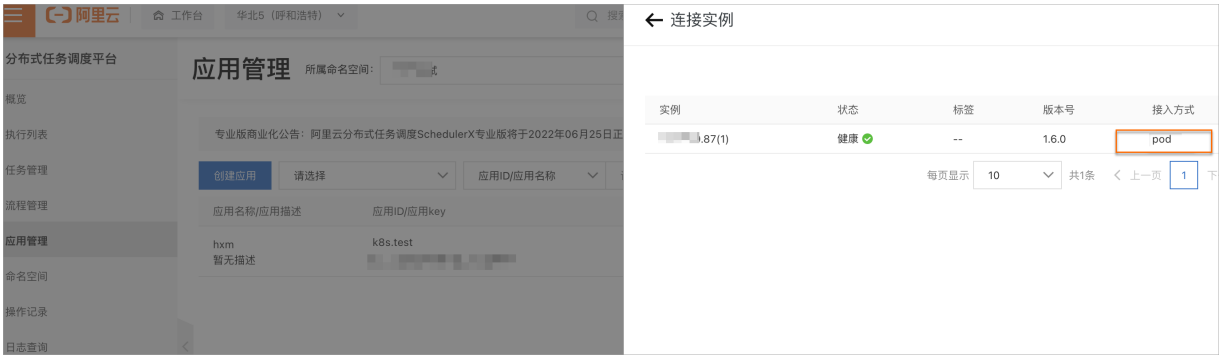
芯片架构	区域	说明
x86_64	公网	registry.cn-hangzhou.aliyuncs.com/schedulerx/agent:1.6.0-amd64
	阿里云VPC	registry-vpc.{regionId}.aliyuncs.com/schedulerx/agent:1.6.0-amd64

芯片架构	区域	说明
arm64	公网	registry.cn-hangzhou.aliyuncs.com/schedulerx/agent:1.6.0-arm64
	阿里云VPC	registry-vpc.{regionId}.aliyuncs.com/schedulerx/agent:1.6.0-arm64

SchedulerX agent env变量说明：

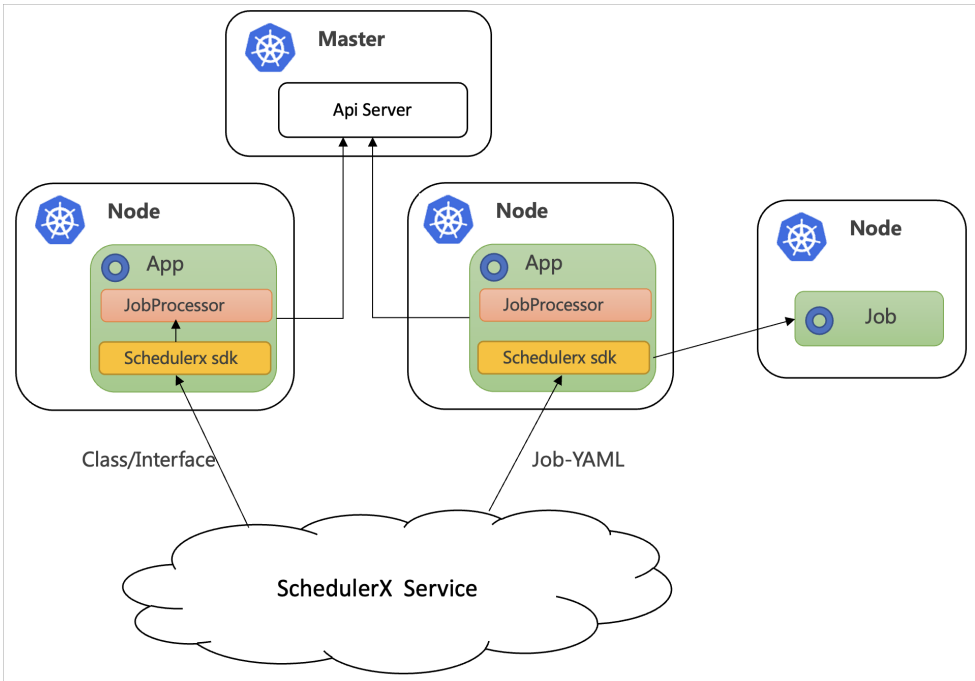
变量	说明
${SCHEDULERX_ENDPOINT}	您部署应用的地域（Region）和对应的Endpoint。比如addr-sh-internal.edas.aliyun.com。详情请参见Endpoint列表。
${SCHEDULERX_NAMESPACE}	Namespace为命名空间ID，可以在SchedulerX控制台的**命名空间**页面获取。 
${SCHEDULERX_GROUPID}	GroupId为应用ID，可以在SchedulerX控制台**应用管理**页面获取。 
${SCHEDULERX_APPKEY}	AppKey为应用Key，可以在SchedulerX控制台**应用管理**页面获取。 

如果部署Deployment完成，可以在SchedulerX控制台**应用管理**页面查看实例，代表接入成功。



通过Java SDK部署

如果您是Java应用，除了想调度K8s任务，还想调度您的Java程序，可以依赖Java SDK。与使用MQ相同，SchedulerX和您的在线业务在一个进程中，原理如下：



关于SDK接入，请参见Spring Boot应用接入SchedulerX。

使用K8s任务还需要依赖一个schedulerx-plugin-kubernetes，比如：

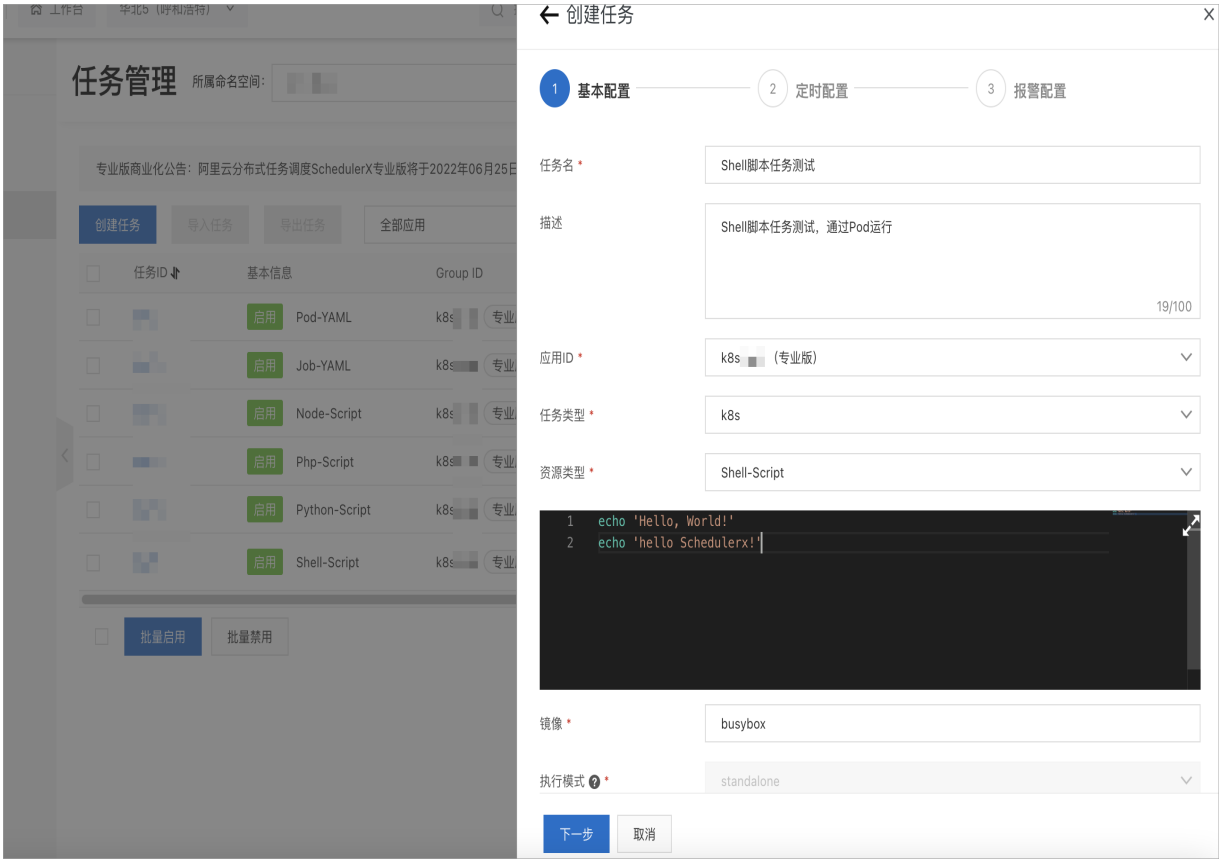
```
<dependency>
        <groupId>com.alibaba.schedulerx</groupId>
        <artifactId>schedulerx2-spring-boot-starter</artifactId>
        <version>1.6.0</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba.schedulerx</groupId>
        <artifactId>schedulerx-plugin-kubernetes</artifactId>
        <version>1.0.1</version>
    </dependency>
```

部署您应用的Deployment文件，还需要配置serviceAccountName: schedulerx（参考上文非Java应用部署的schedulerx-agent.yaml）。

步骤三：创建K8s任务

Shell脚本

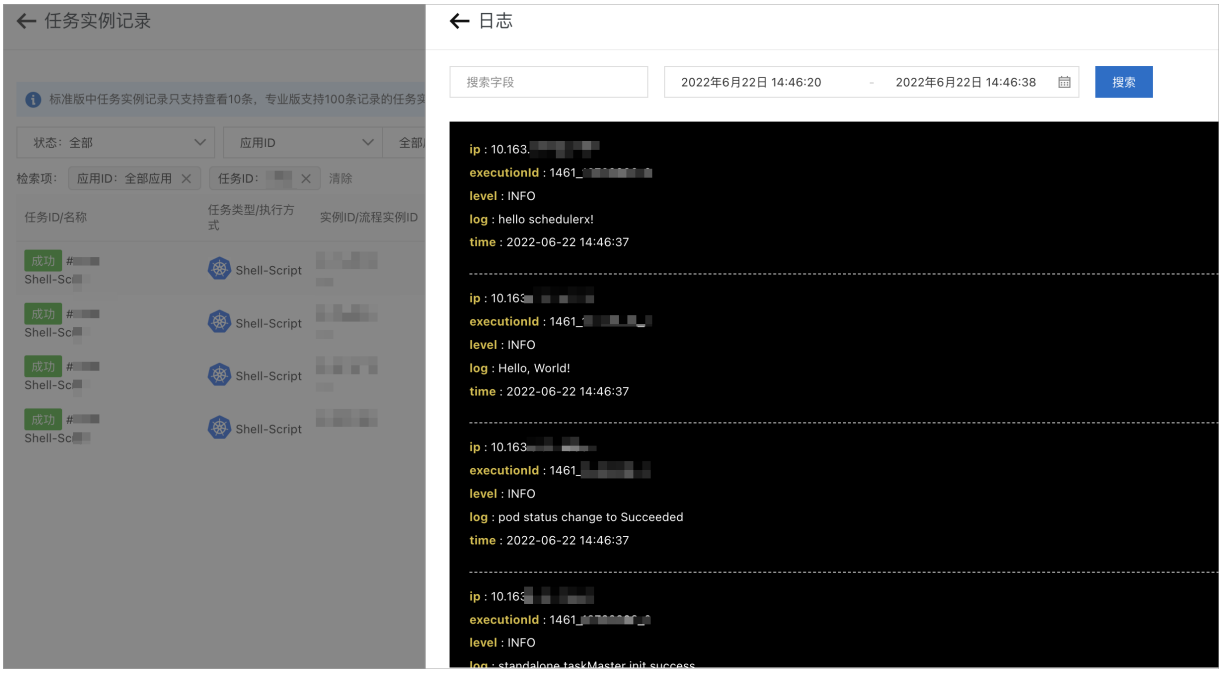
如果想通过Pod运行Shell脚本，不需要自己构建镜像，只需要在任务管理创建一个K8s任务，资源类型选择Shell-Script，镜像默认是busybox（也可以替换为自己的镜像）。



单击**运行一次**，在Kubernetes集群中可以看到Pod启动，Pod名称为schedulerx-shell-{JobId}。

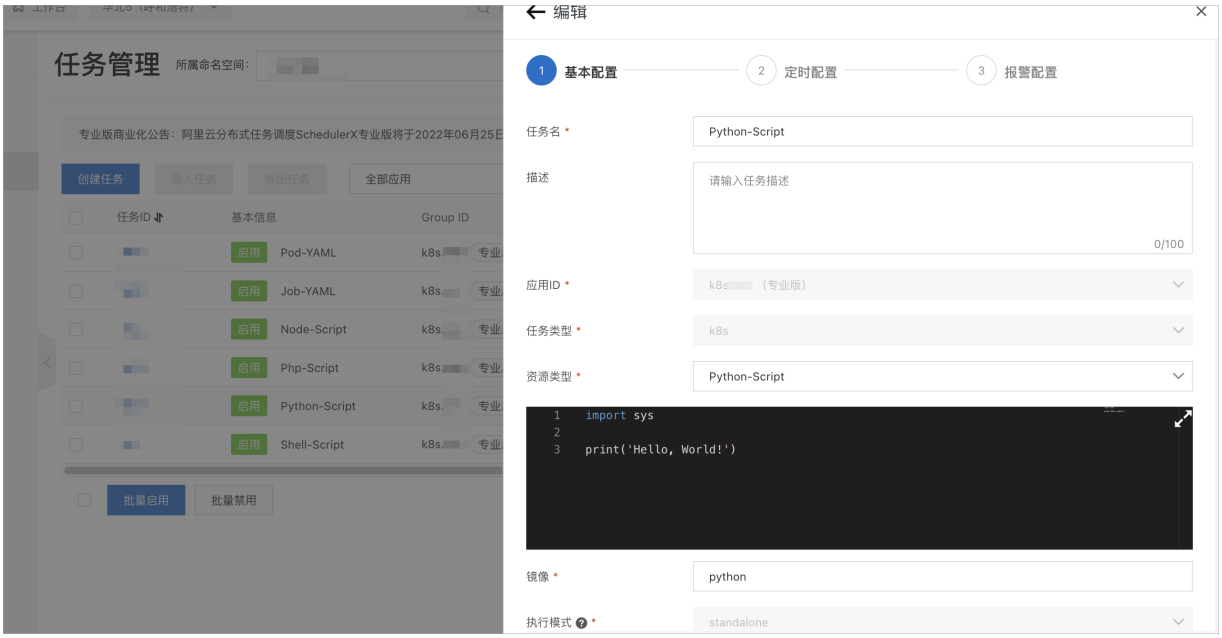
```
[root@schedulerx-agent-                                    ~]
# kubectl get pod | grep schedulerx
schedulerx-agent-69d448d974-xzjw6   1/1     Running     0          18h
schedulerx-shell-1461               0/1     Completed   0          2m37s

[root@schedulerx-agent-                                    ~]
# kubectl logs schedulerx-shell-1461
Hello, World!
hello schedulerx!

[root@schedulerx-agent-                                    ~]
#
```

在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志。



Python脚本

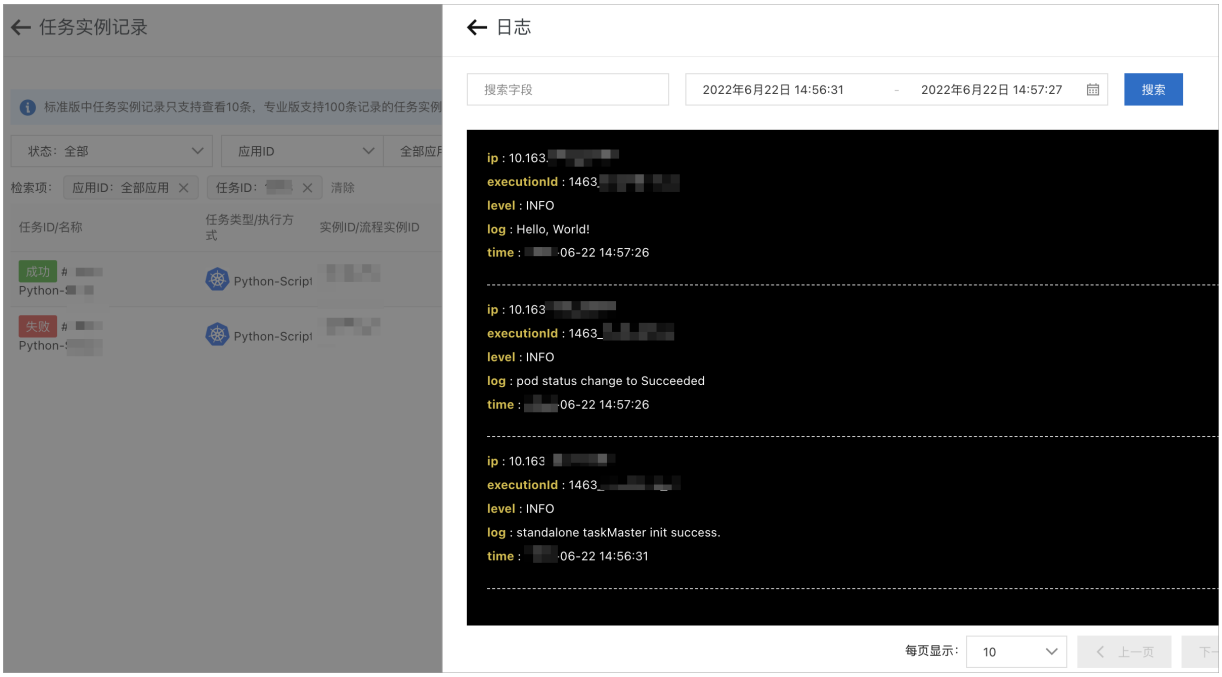
如果想通过Pod运行Python脚本，不需要自己构建镜像，只需要在任务管理创建一个K8s任务，资源类型选择Python-Script，镜像默认是Python（也可以替换为自己的镜像）。



单击**运行一次**，在Kubernetes集群中可以看到Pod启动，Pod名称为schedulerx-python-{JobId}。

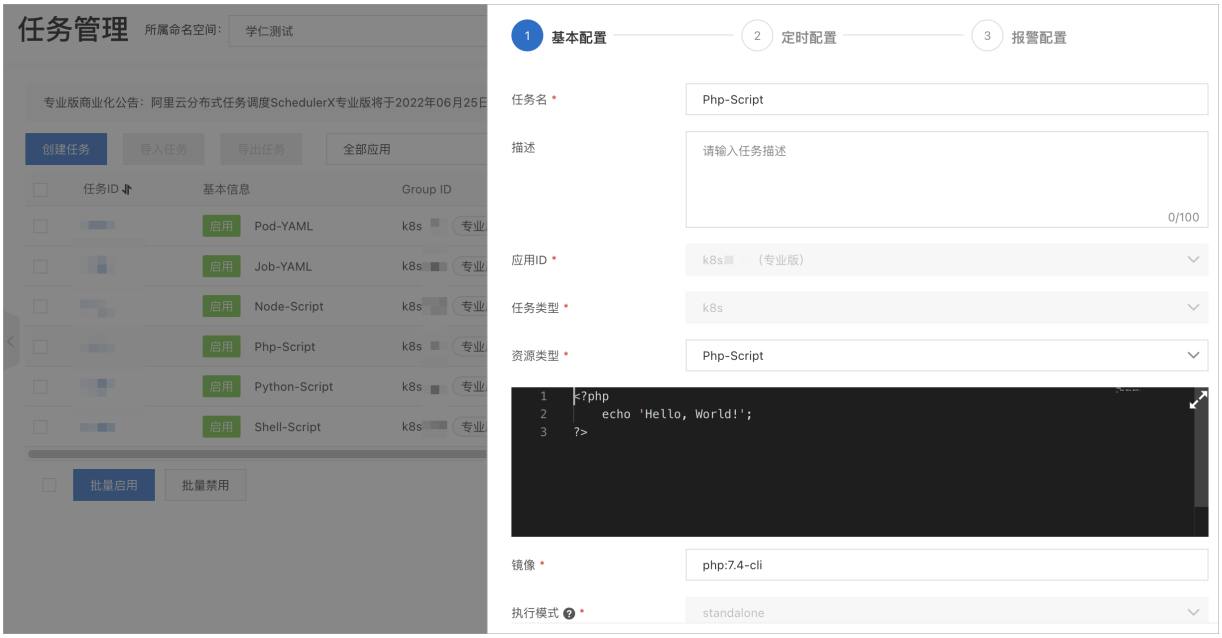
```
[root@schedulerx-agent [redacted] /]
# kubectl get pod | grep schedulerx
schedulerx-agent-69d448d974-xzjw6   1/1     Running     0          18h
schedulerx-python-1463              0/1     Completed   0          5m37s
schedulerx-shell-1461               0/1     Completed   0          15m

[root@schedulerx-agent[redacted] /]
# kubectl logs schedulerx-python-1463
Hello, World!

[root@schedulerx-agent-6[redacted] /]
#
```

在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志。



PHP脚本

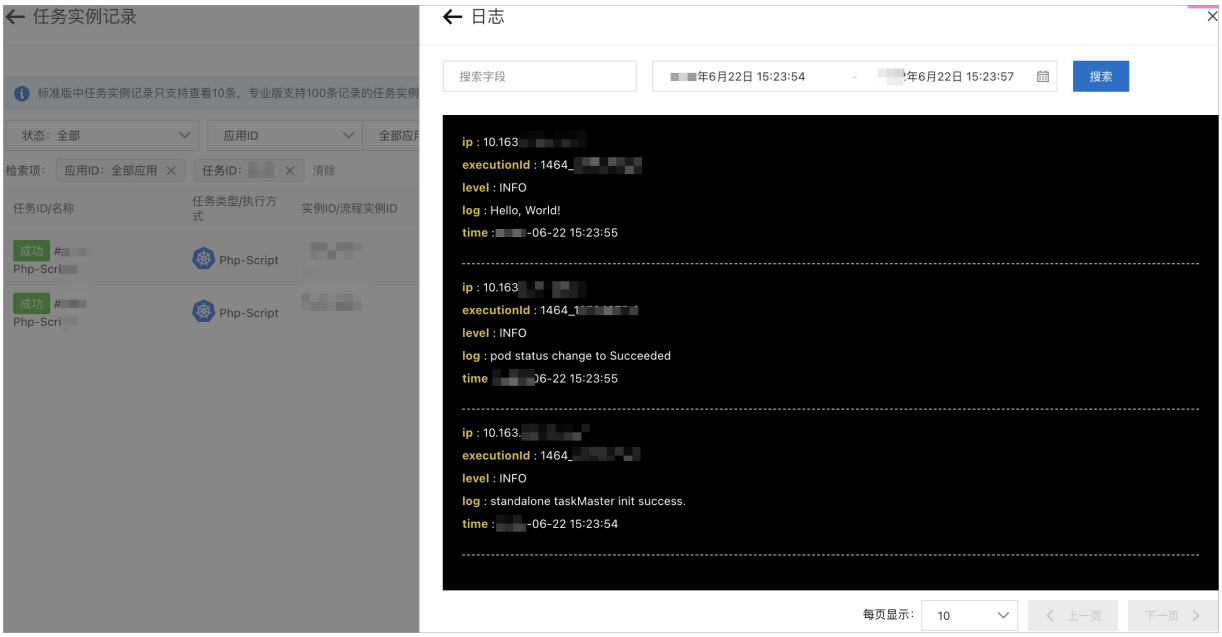
如果想通过Pod运行PHP脚本，不需要自己构建镜像，只需要在任务管理创建一个K8s任务，资源类型选择Php-Script，镜像默认是php:7.4-cli（也可以替换自己的镜像）。



单击**运行一次**，在Kubernetes集群中可以看到Pod启动，Pod名称为schedulerx-php-{JobId}。

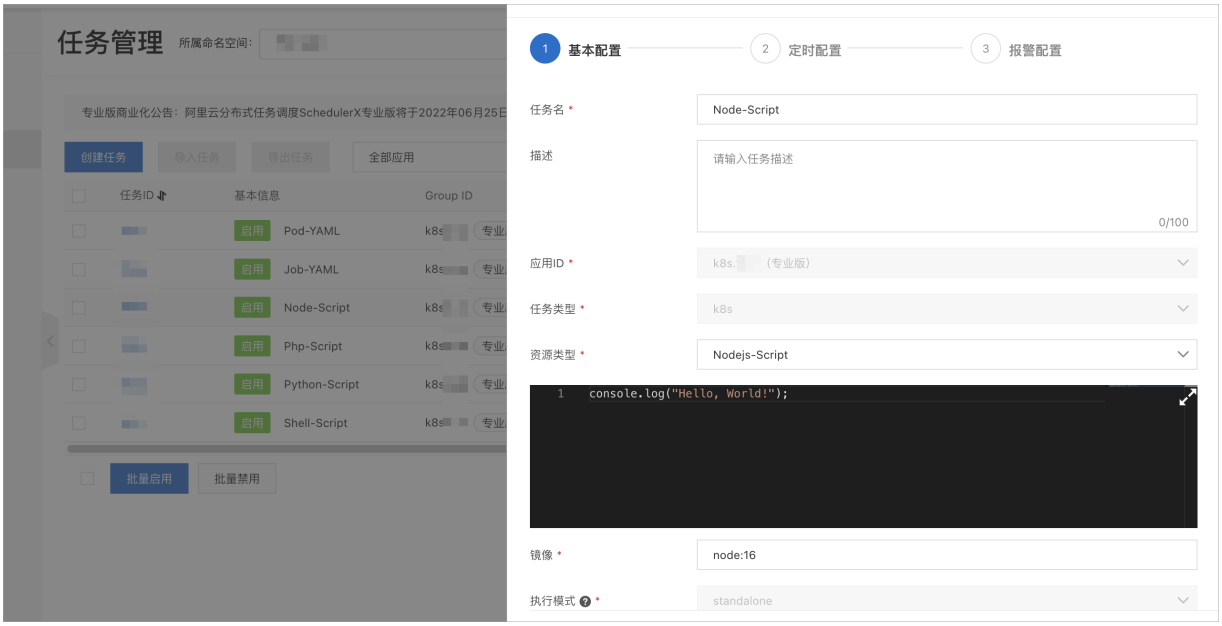
```
[root@schedulerx-agent
# kubectl get pod | grep schedulerx
schedulerx-agent-                    1/1     Running     0          19h
schedulerx-php-1464                  0/1     Completed   0          54s
schedulerx-python-1463               0/1     Completed   0          28m
schedulerx-shell-1461                0/1     Completed   0          38m

[root@schedulerx-agent
# kubectl logs schedulerx-
Hello, World!
[root@schedulerx-agent-
#
```

在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志

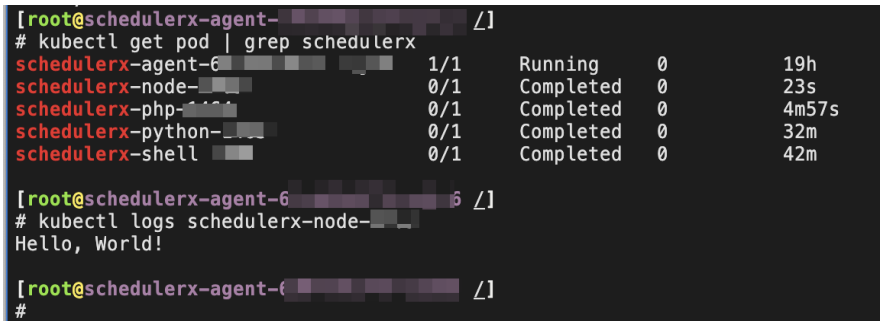


Node.js脚本

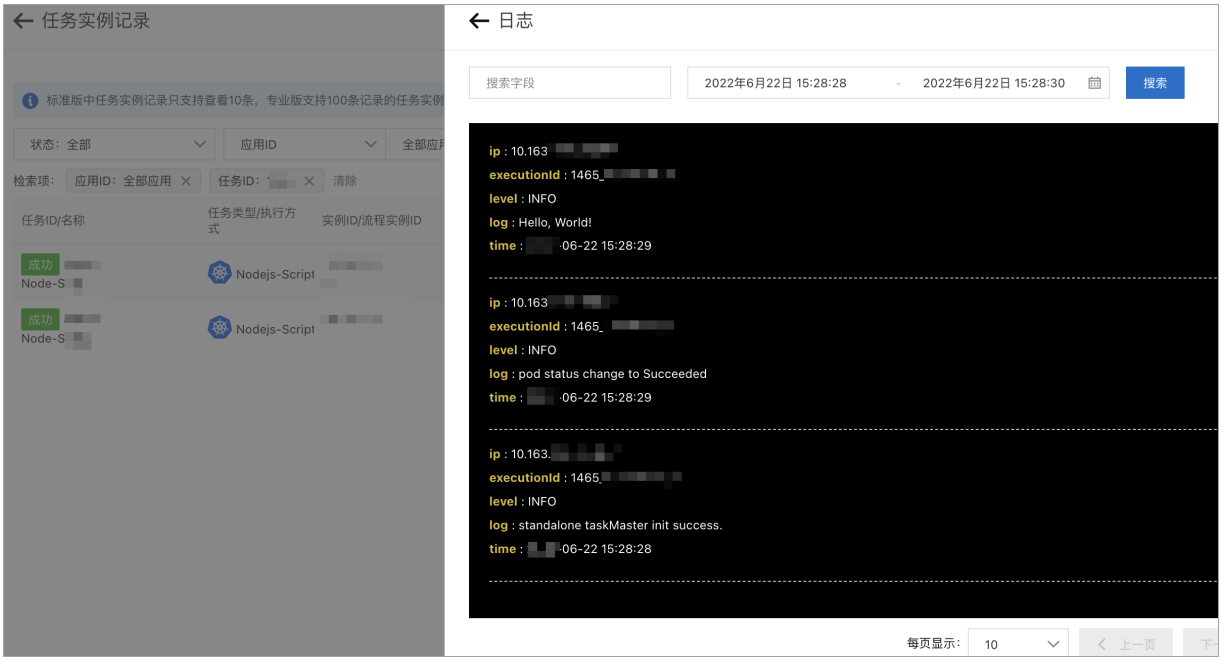
如果想通过Pod运行Node.js脚本，不需要自己构建镜像，只需要在任务管理创建一个K8s任务，资源类型选择Nodejs-Script，镜像默认是node:16（也可以替换自己的镜像）。



单击**运行一次**，在Kubernetes集群中可以看到Pod启动，Pod名称为schedulerx-node-{JobId}。

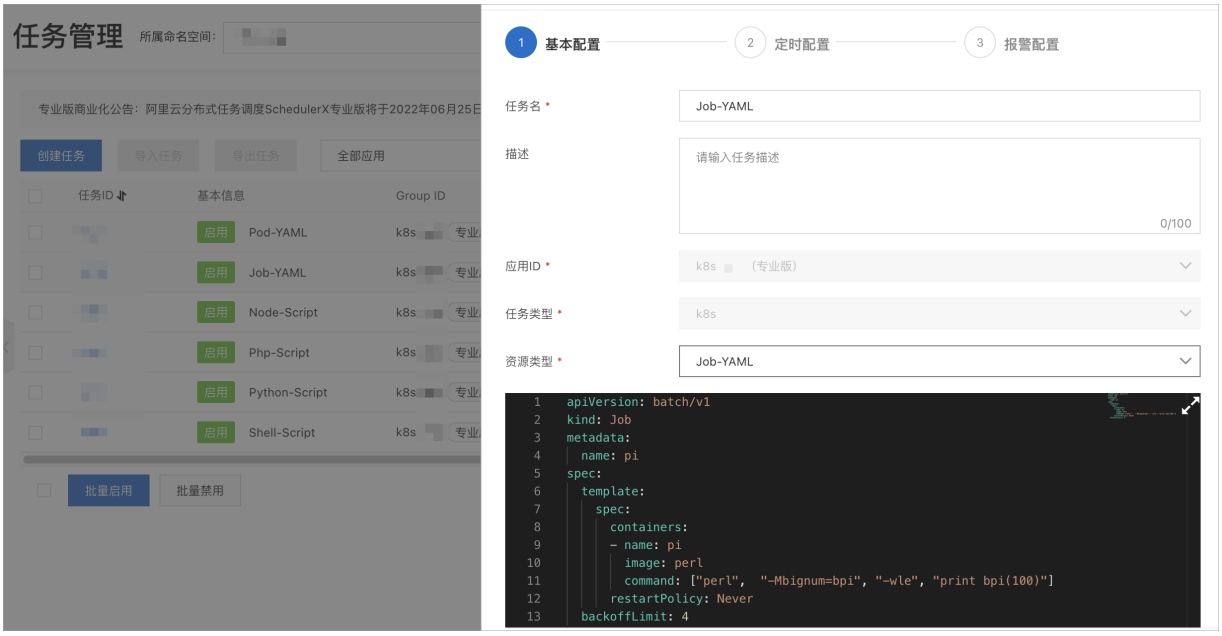


在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志。



Job-YAML

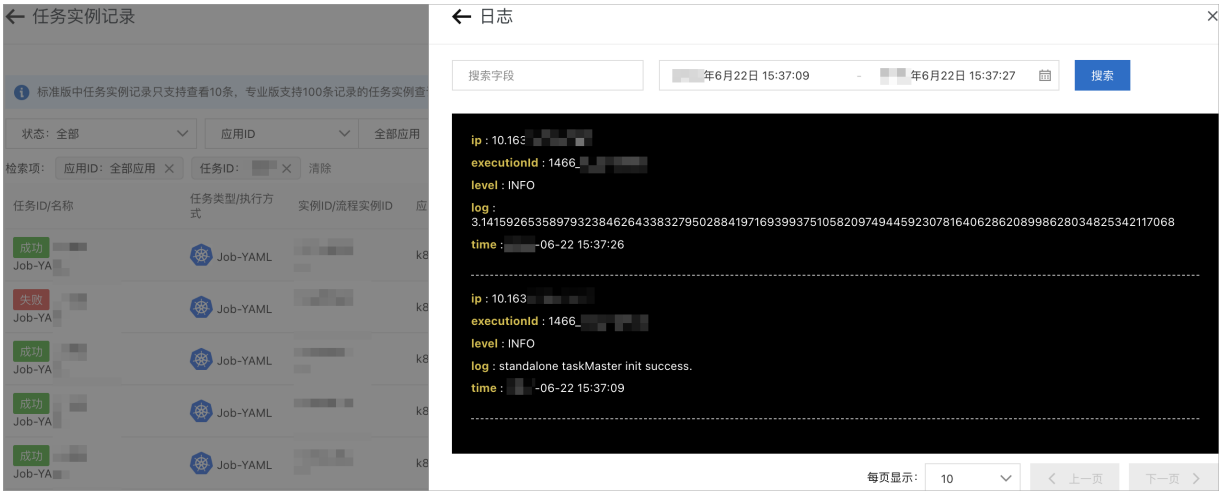
通过SchedulerX也可以运行K8s原生的Job，任务类型选择K8s，资源类型选择Job-YAML。



单击**运行一次**，在Kubernetes集群中可以看到Job和Pod启动成功。

```
[root@schedulerx-agent ~]
# kubectl get job
NAME   COMPLETIONS   DURATION   AGE
pi     1/1           17s        6m5s

[root@schedulerx-agent ~]
# kubectl get pod
NAME                          READY   STATUS      RESTARTS   AGE
hello                         0/1     Completed   0          19h
pi--1-jgck5                   0/1     Completed   0          6m8s
schedulerx-agent-             1/1     Running     0          19h
schedulerx-node-1             0/1     Completed   0          14m
schedulerx-php-14             0/1     Completed   0          19m
schedulerx-python             0/1     Completed   0          46m
schedulerx-shell-             0/1     Completed   0          56m

[root@schedulerx-agent- ~]
# kubectl logs pi--1-jgck5
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 8

[root@schedulerx-agent- ~]
#
```

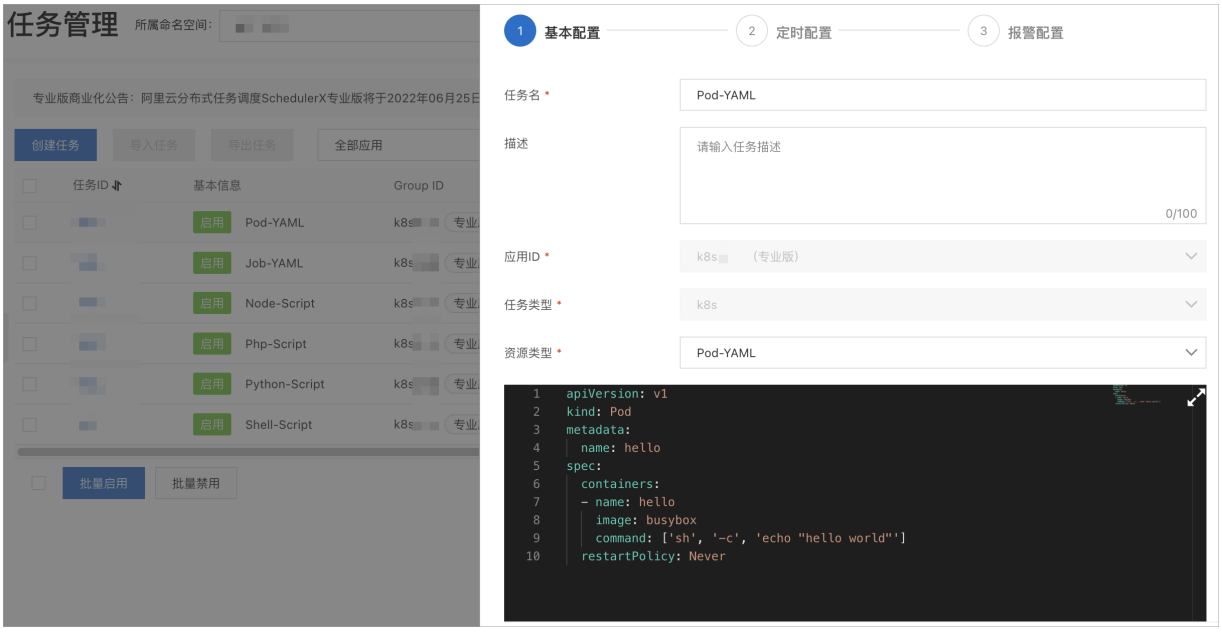
在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志。



> 注意　通过SchedulerX运行K8s Job，不建议使用CronJob，定时调度需要使用SchedulerX来配置，否则无法收集每次Pod的执行历史和日志。

Pod-YAML

通过SchedulerX也可以运行K8s原生的Pod，任务类型选择K8s，资源类型选择Pod-YAML。



单击**运行一次**，在Kubernetes集群中可以看到Pod启动成功。

```
[root@schedulerx-agent-          /]
# kubectl get pod
NAME                                READY   STATUS      RESTARTS   AGE
hello                               0/1     Completed   0          2m39s
pi--1-jgck5                         0/1     Completed   0          49m
schedulerx-agent-                   1/1     Running     0          20h
schedulerx-node-                    0/1     Completed   0          58m
schedulerx-php-1                    0/1     Completed   0          62m
schedulerx-pytho                    0/1     Completed   0          90m
schedulerx-shell                    0/1     Completed   0          100m

[root@schedulerx-                   /]
# kubectl logs hello
hello world

[root@schedulerx-agent-             /]
#
```

在SchedulerX控制台**任务管理**页面可以查询历史执行记录，也可以看到Pod运行的日志。

> **注意** 通过SchedulerX运行K8s Pod，建议不要运行长周期的Pod（比如Web应用，一旦启动永远不会结束），重启策略需要设置成Never（否则Pod会不断重启）。

6.4.2.8. Endpoint列表

初始化SchedulerxWorker时，会用到您部署应用的地域（Region）和对应的Endpoint。

地域（Region）	Endpoint	用途
华东1（杭州）	addr-hz-internal.edas.aliyun.com	线上生产环境
华东2（上海）	addr-sh-internal.edas.aliyun.com	线上生产环境
华北2（北京）	addr-bj-internal.edas.aliyun.com	线上生产环境
华北3（张家口）	addr-cn-zhangjiakou-internal.edas.aliyun.com	线上生产环境
华北5（呼和浩特）	addr-cn-huhehaote-internal.edas.aliyun.com	线上生产环境
华南1（深圳）	addr-sz-internal.edas.aliyun.com	线上生产环境
中国香港	addr-hk-internal.edas.aliyuncs.com	线上生产环境
新加坡（新加坡）	addr-singapore-internal.edas.aliyun.com	线上生产环境
华北2政务云1	addr-cn-north-2-gov-1-internal.edas.aliyun.com	线上生产环境
上海金融云	addr-cn-shanghai-finance-1-internal.edas.aliyun.com	线上生产环境
美国（弗吉尼亚）	addr-us-east-1-internal.acm.aliyun.com	线上生产环境
澳大利亚（悉尼）	addr-ap-southeast-2-internal.edas.aliyun.com	线上生产环境

地域（Region）	Endpoint	用途
德国（法兰克福）	addr-eu-central-1-internal.edas.aliyun.com	线上生产环境
日本（东京）	addr-ap-northeast-1-internal.edas.aliyun.com	线上生产环境
公网	acm.aliyun.com	线上生产环境

6.5. Scheduling和Triggers

6.5.1. 定时调度

6.5.1.1. Cron

Cron是一款类Unix的操作系统下的基于时间的任务管理系统。您可以通过Cron在固定时间、日期间隔下运行定时任务（可以是命令和脚本）。

Cron表达式

Cron的表达式为：`秒分时 日月周[年]`

为了帮助您理解，下面介绍一些常用的Cron表达式示例。

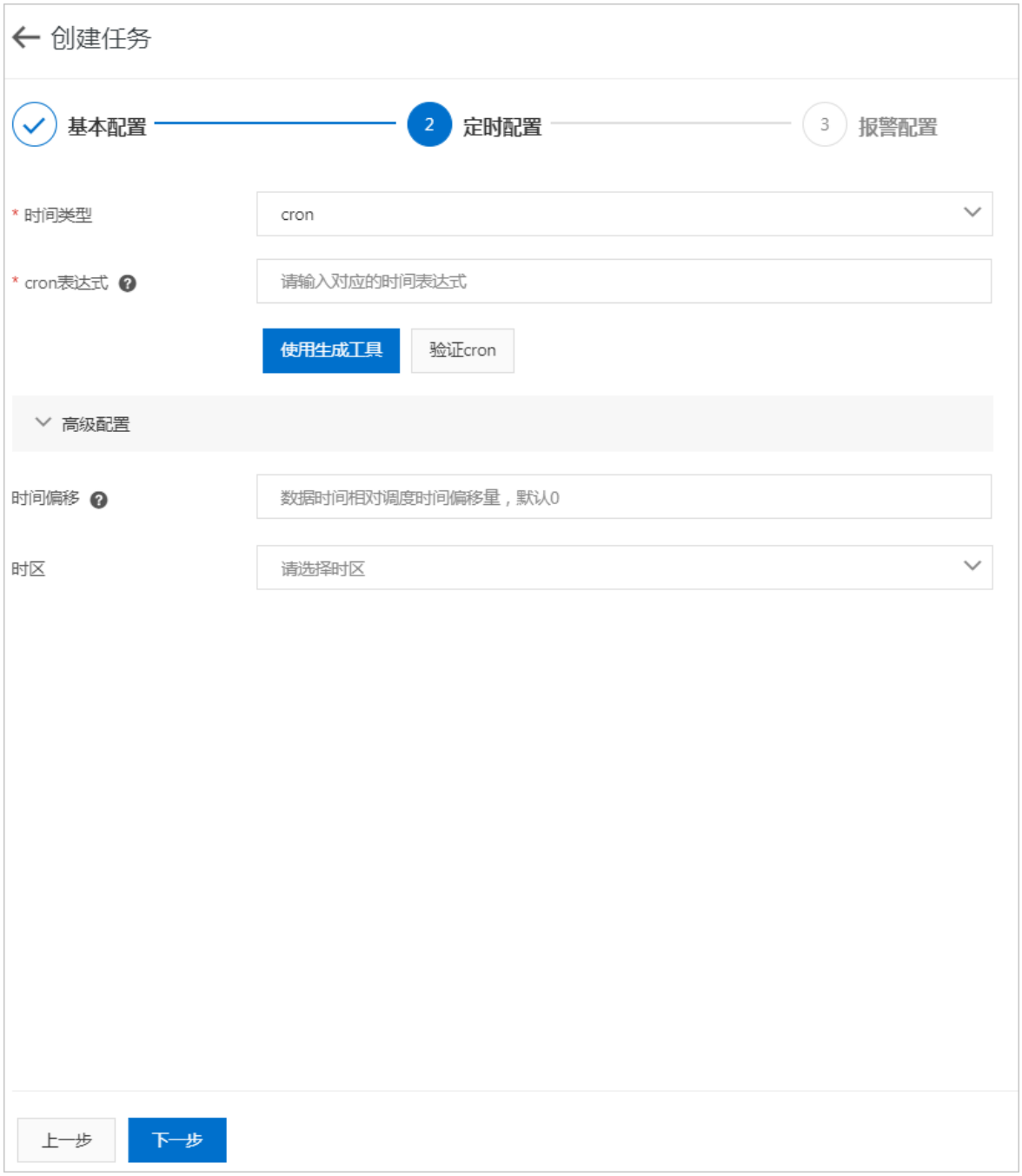
Contab表达式	说明
0 */1 * * * ?	每隔1分钟触发一次
0 0 5-15 * * ?	每天5:00~15:00整点触发
0 0/3 * * * ?	每隔3分钟触发一次
0 0-5 14 * * ?	每天14:00~14:05期间每隔1分钟触发一次
0 0/5 14 * * ?	每天14:00~14:55期间每隔5分钟触发一次
0 0/5 14,18 * * ?	每天14:00~14:55和18:00~18:55两个时间段内每5分钟触发一次
0 0/30 9-17 * * ?	每天9:00~17:00内每半小时触发一次
0 0 10,14,16 * * ?	每天10:00、14:00和16:00触发
0 0 12 ? * WED	每周三12:00触发
0 0 17 ? * TUES,THUR,SAT	每周二、周四、周六17:00触发
0 10,44 14 ? 3 WED	每年3月的每周三的14:10和14:44触发
0 15 10 ? * MON-FRI	周一至周五的上午10:15触发
0 0 23 L * ?	每月最后一天23:00触发
0 15 10 L * ?	每月最后一天10:15触发
0 15 10 ? * 6L	每月最后一个周五10:15触发

Contab表达式	说明
0 15 10 * * ? 2005	2005年的每天10:15触发
0 15 10 ? * 6L 2002-2005	2002年~2005年的每月的最后一个周五上午10:15触发
0 15 10 ? * 6#3	每月的第三个周五10:15触发

Cron定时调度配置示例

在创建调度任务时可以使用Cron定时调度，创建调度任务的详细操作步骤请参见创建调度任务。本文仅介绍在创建调度任务时如何配置Cron定时调度。

1. 进入**定时配置**页签，详情请参见创建调度任务。
2. 在**创建任务**配置向导的**定时配置**页签中设置定时调度参数，然后单击**下一步**。



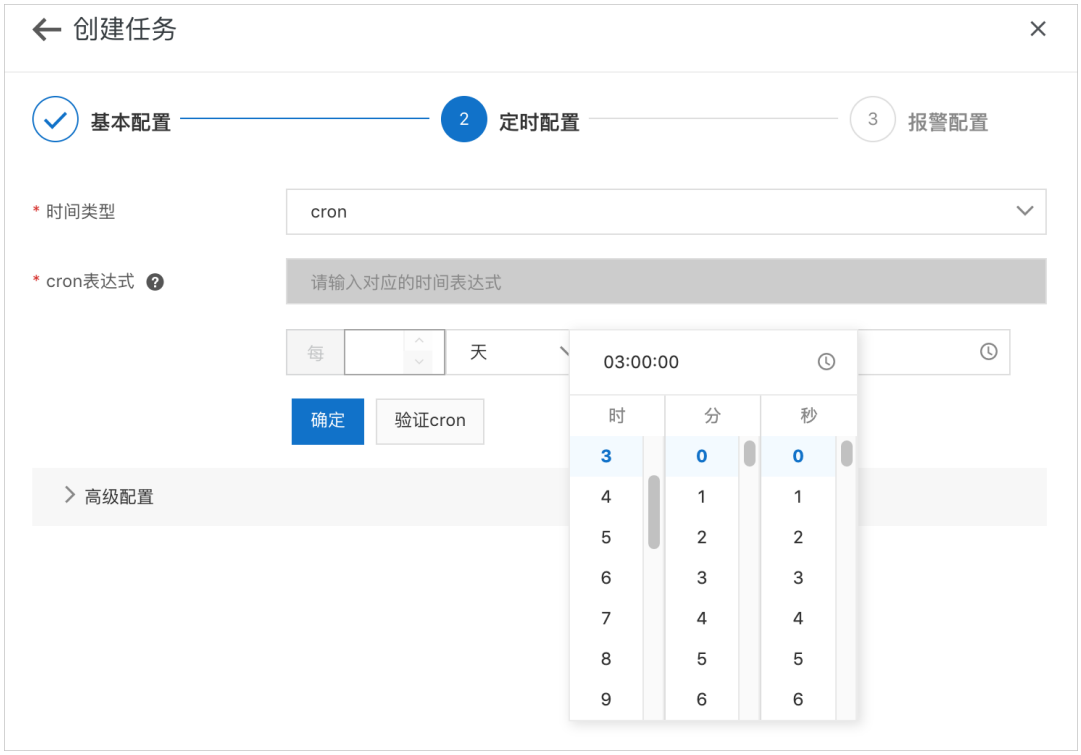
i. 在**时间类型**右侧的列表选择**cron**。

ii. 设置**cron表达式**。

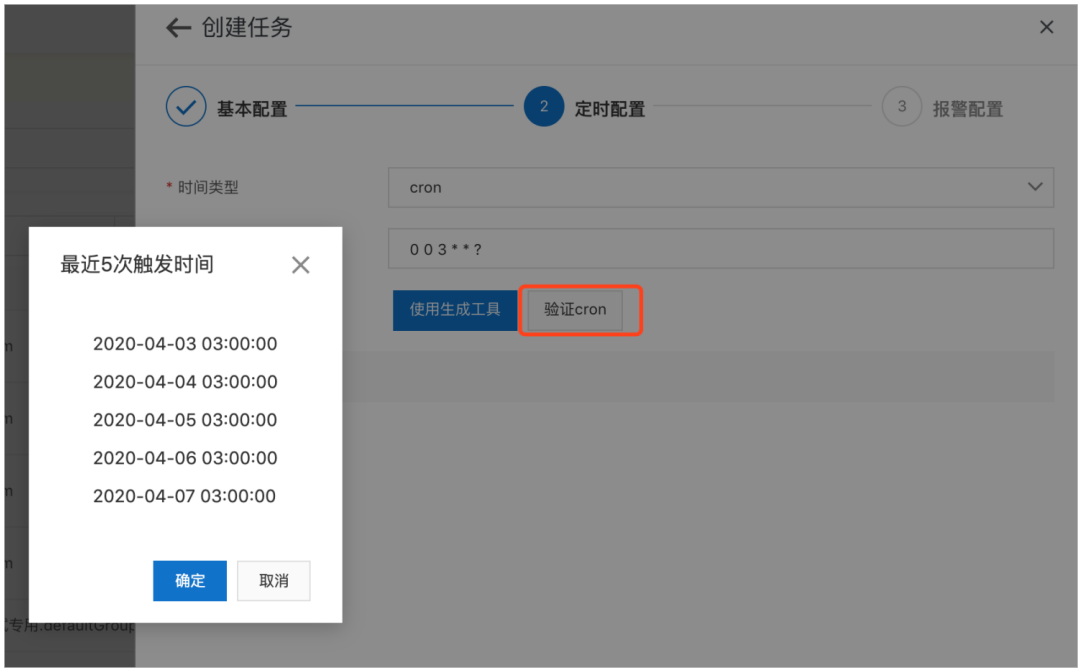
如果定时规则比较简单，建议使用工具生成，本文也会介绍如何使用工具生成Cron 表达式。如果规则复杂，工具无法生成，可以输入Cron表达式，相关示例请参见Cron表达式。

a. 单击**使用生成工具**。

b. 在弹出的列表中选择时间周期、时间等规则，例如每天3:00执行，然后单击**确定**。



c. 单击**验证cron**，查看规则最近5次触发时间。



iii. （可选）如果需要，在**高级设置**区域还可以设置**时间偏移**和**时区**。

当时间类型选择Cron后，可以进行高级配置。高级配置参数说明如下：

配置名称	意义
时间偏移	数据时间相对于调度时间的偏移，可以在调度时从上下文获取该值。
时区	可以根据实际情况选择不同时区，包括一些常用国家或地区，也包括标准的GMT表达方式。

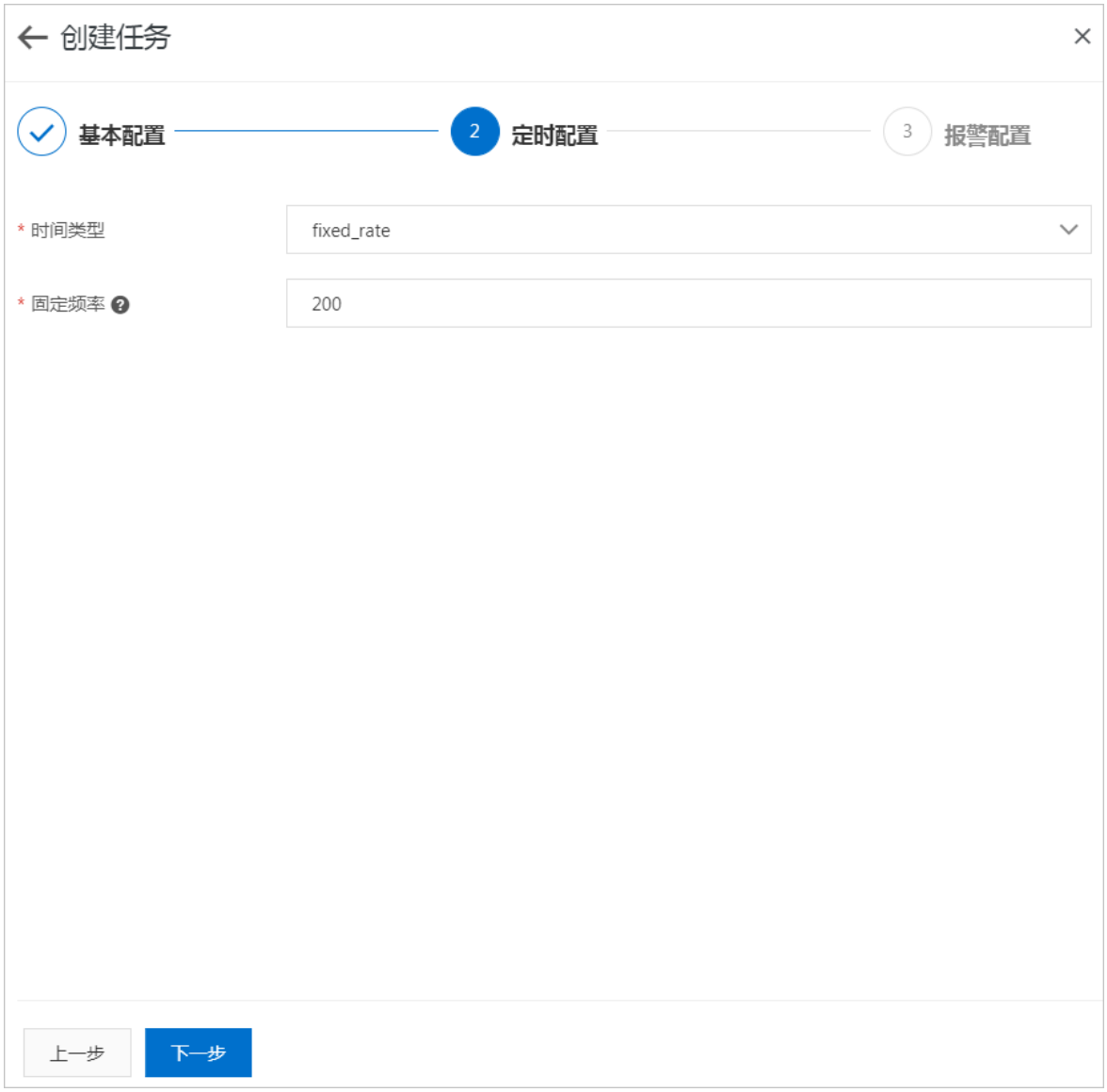
6.5.1.2. Fixed rate

由于Crontab必须被60整除，如果需要每隔40分钟执行一次调度，则Cron无法支持。Fixed rate专门用来做定期轮询，可以解决该问题，且表达式简单，但不支持秒级别。

Fixed rate定时配置示例

在创建调度任务时可以使用Fixed rate定时调度，创建调度任务的详细操作步骤请参见创建调度任务。本文仅介绍在创建调度任务时如何配置Fixed rate定时调度。

1. 进入**定时配置**页签，详情请参见创建调度任务。
2. 在**创建任务**配置向导的**定时配置**页签中设置定时调度参数，然后单击**下一步**。



i. 在**时间类型**右侧的列表中选择**fixed_rate**。

ii. 在**固定频率**右侧输入执行间隔。

> **说明** 固定频率必须高于60秒。

6.5.1.3. Second delay

Second delay即秒级别调度，适用于对实时性要求比较高的业务，例如需要不停做轮询的准实时业务。

背景信息

Second delay支持1~60秒间隔的秒级延迟调度，即每次任务执行完成后，间隔秒级时间再次触发调度。

Second delay具有以下优势：

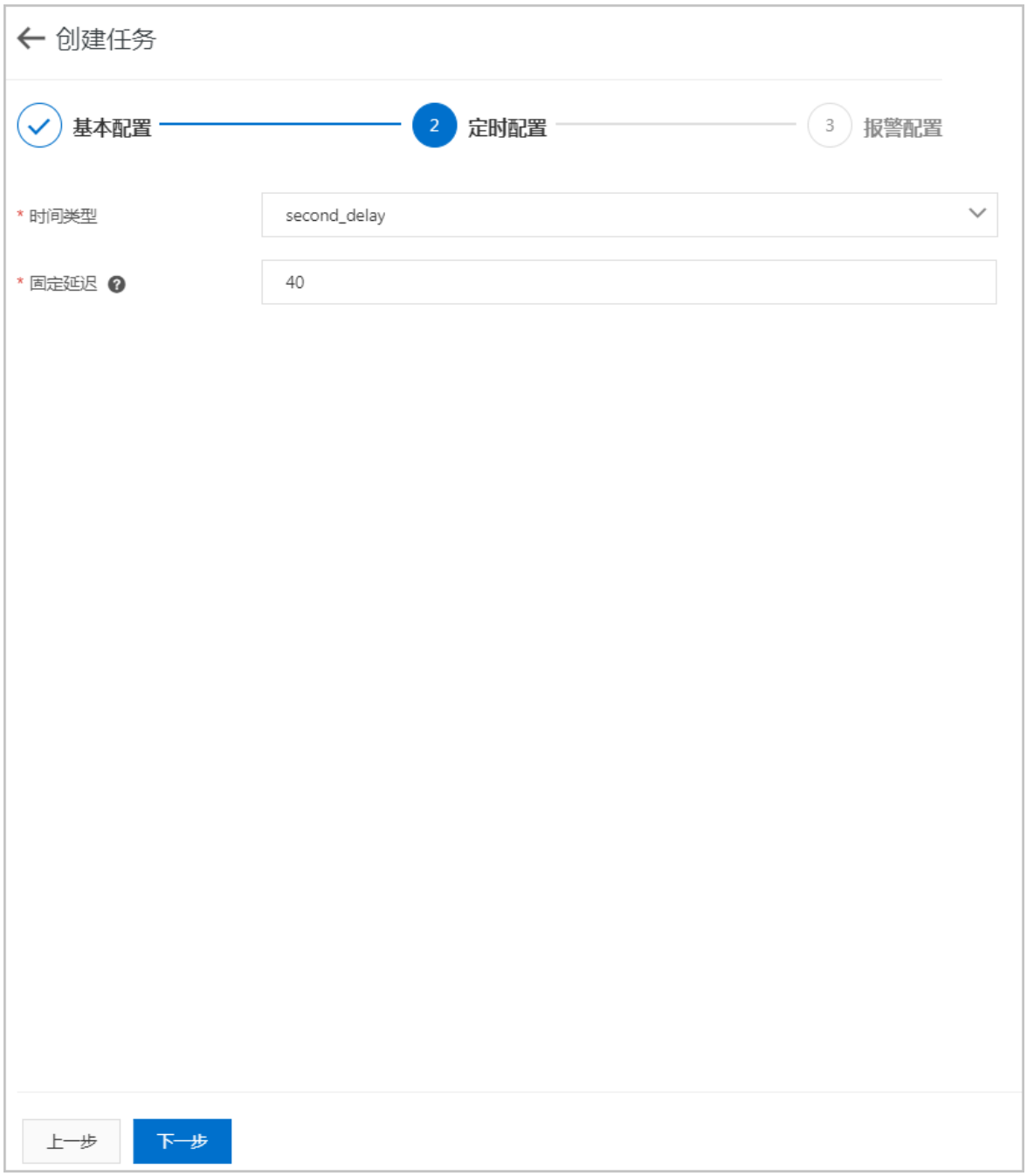
- 高可靠：SchedulerX的秒级别任务具有高可靠的特性，如果某台机器宕机了，可以在30秒内在另一台机器上重新拉起。
- 丰富的任务类型：SchedulerX的秒级别任务属于定时调度类型，可以适用于所有的任务类型和执行方式。不但简单Java任务支持秒级别调度，分布式Java任务以及脚本任务同样适用。通过内存网格和秒级别调度，可以不停地处理海量的数据。

- 统计信息：SchedulerX还提供了秒级别任务的统计信息，例如当天执行了多少次，成功的次数和失败的次数，最近10次运行结果等。

Second delay定时配置示例

在创建调度任务时可以使用Second delay定时调度，创建调度任务的详细操作步骤请参见创建调度任务。本文仅介绍在创建调度任务时如何配置Second delay定时调度。

1. 进入**定时配置**页签，详情请参见创建调度任务。
2. 在**创建任务**配置向导的**定时配置**页签中设置定时调度参数，然后单击**下一步**。



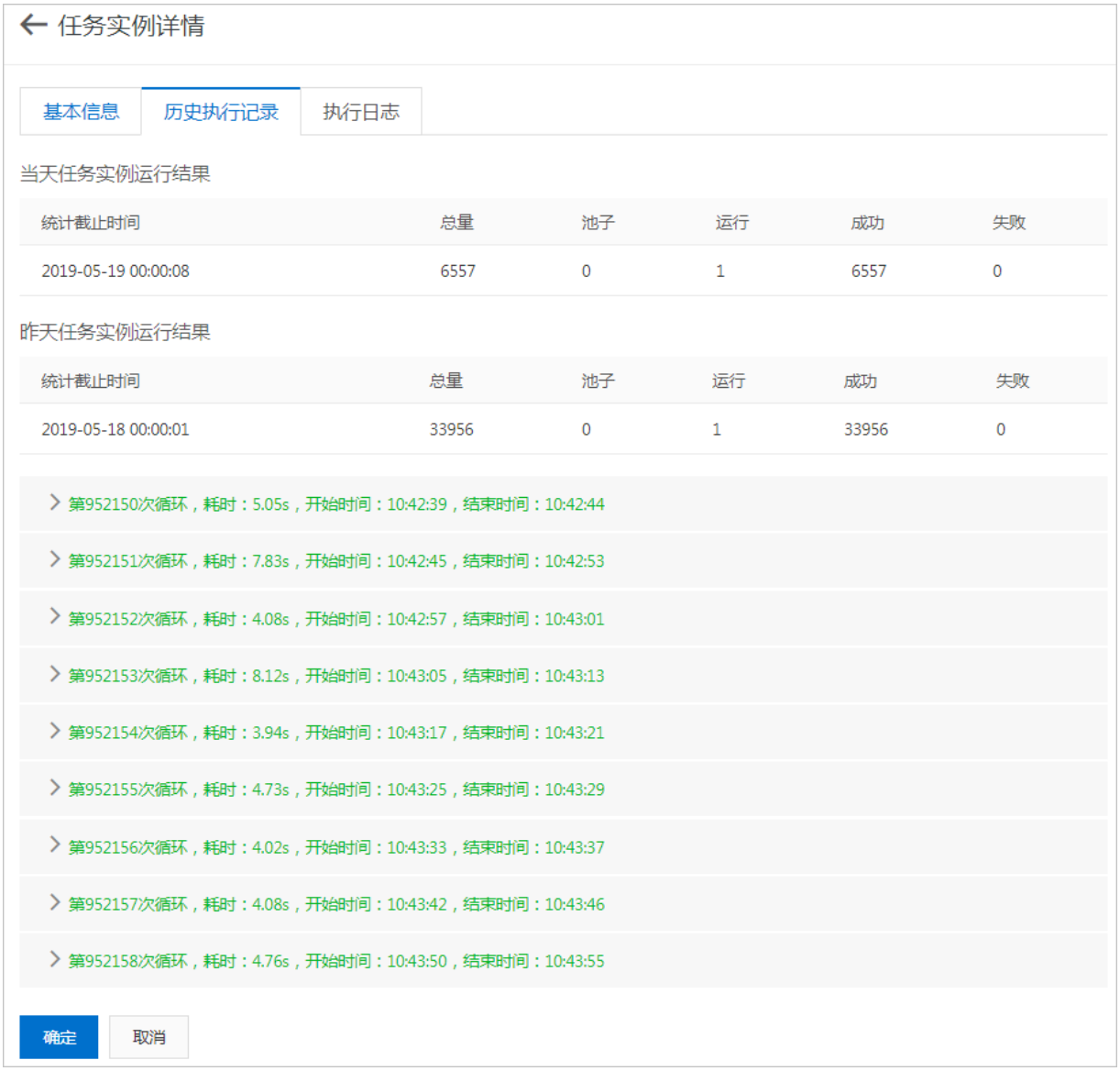
i. 在**时间类型**右侧的列表中选择**second_delay**。

ii. 在**固定延迟**右侧输入固定延迟，单位为秒，范围为1秒~60秒。

查看秒级别任务的统计信息

Second delay任务创建完成后，在**任务实例详情**页面会多一个**秒级任务统计详情**页签，展示如下信息：

- 当天任务实例运行结果
- 昨天任务实例运行结果
- 最近10次运行结果



6.5.1.4. One Time

One Time即一次性任务调度，配置完成后任务将在设置的时间点执行一次，执行完成后调度平台会自动清理任务，您无需在客户端执行删除任务的操作。一次性任务调度适用于订单超时未支付、自动关闭定时日历提醒等场景。

优势

- **精准时刻**：SchedulerX的一次性任务与延时消息相比，没有固定延迟多少时间或者在多少天内的时间限制，支持未来任意时间点，使用简单。
- **丰富的任务类型**：SchedulerX一次性任务适用于所有任务类型，例如Java、HTTP、Shell任务。也适用所有分布式模型，例如单机、广播、分片、MapReduce等。
- **可视化运维**：SchedulerX的一次性任务和其他任务一样，具有可视化的界面，方便您观测和查询。并且支持在调度时间到达之前修改任务参数，支持失败自动重试等功能。

通过API创建一次性调度任务

1. 在应用程序的*pom.xml*文件中添加OpenAPI的SDK依赖。具体操作，请参见添加SDK依赖。
2. 下面以一个示例代码说明如何使用SDK调用API，将timeType的参数值设置为5，表示启用一次性任务调度。

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobResponse;
public class CreateJavaJob {
    public static void main(String[] args) throws Exception {
        // OpenAPI的接入点，可查看产品支持的地域列表或根据购买产品的地域填写
        String regionId = "public";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取
        String accessKeyId = "xxxxxxxx";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取
        String accessKeySecret = "xxxxxxxxx";
        //产品名称
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写
        String domain ="schedulerx.aliyuncs.com";
        //构建OpenAPI客户端
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessK
eySecret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        CreateJobRequest request = new CreateJobRequest();
        request.setJobType("java");
        request.setExecuteMode("standalone");
        request.setDescription("test");
        request.setName("一次性任务测试");
        request.setClassName("com.alibaba.schedulerx.test.processor.HelloWorldJob3");
        //timeType=5表示一次性任务
        request.setTimeType(5);
        //yyyy-MM-dd HH:mm:ss
        request.setTimeExpression("2021-12-15 12:11:00");
        request.setNamespace("433d8b23-06e9-408c-aaaa-90d4d1b9a4af");
        request.setGroupId("xueren_sub");
        // 监控报警
        request.setTimeoutEnable(true);
        request.setTimeoutKillEnable(true);
        request.setFailEnable(true);
        request.setTimeout(12300L);
        // 高级配置，配置失败自动重试
        request.setMaxAttempt(3);
        request.setAttemptInterval(30);
        CreateJobResponse response = client.getAcsResponse(request);
        if (response.getSuccess()) {
            System.out.println("jobId=" + response.getData().getJobId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

3. API新建任务后，登录控制台查看任务。
   i. 登录分布式任务调度平台。
   ii. 在左侧导航栏单击**应用管理**。

6.5.2. 工作流调度

通过可视化的工作流进行任务编排，支持Cron表达式和API。

背景信息

工作流中的Job没有独立调度时间，跟随工作流的时间开始调度。

支持上下游数据传输，请参见如何通过工作流进行上下游数据传递。

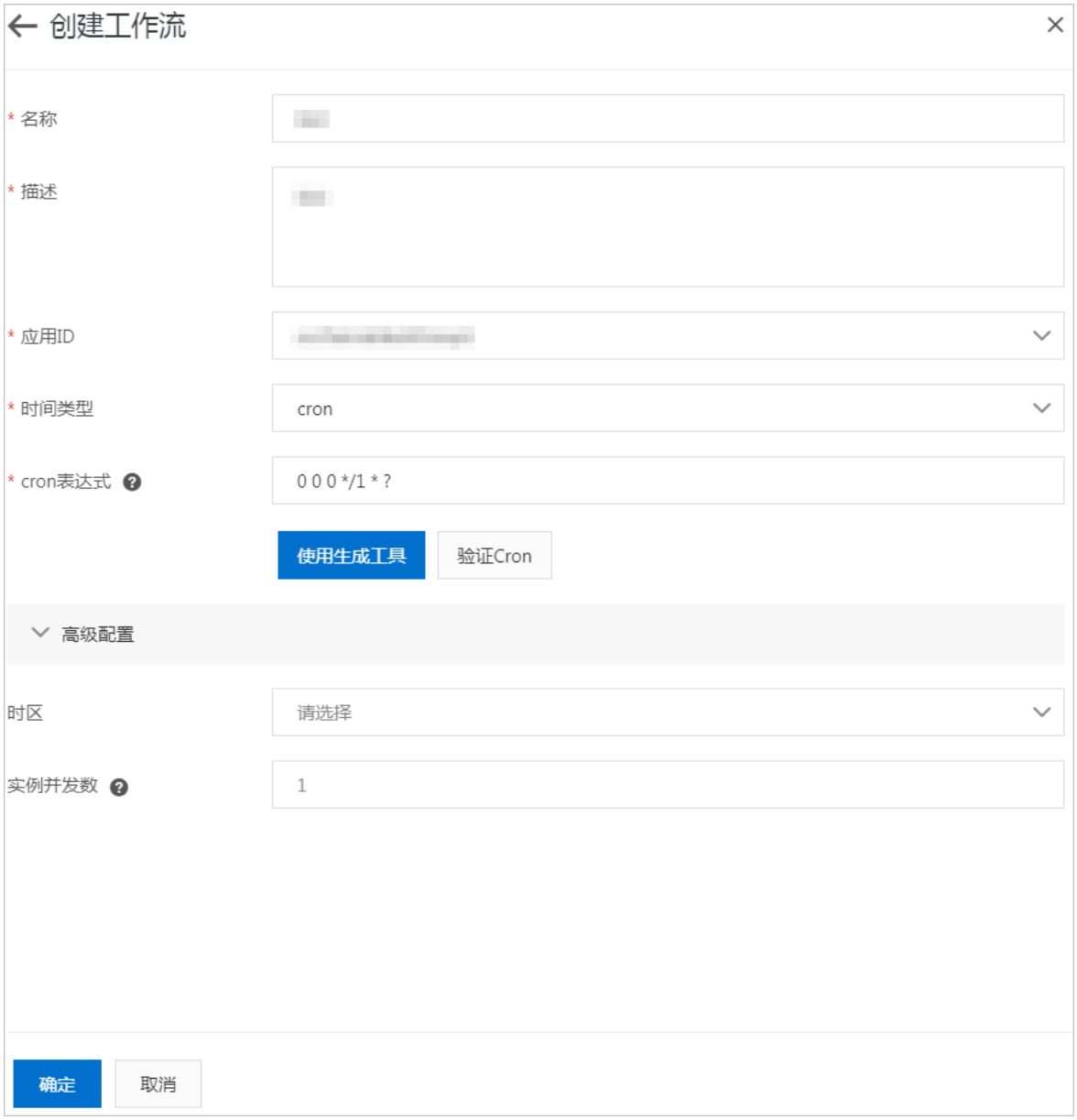
工作流调度至少要有2个Job，且有依赖关系。如果只有一个Job，请直接使用任务管理。

创建工作流

您可以创建工作流调度任务。

> **说明** 目前工作流调度仅支持Cron表达式。

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏单击**流程管理**。
5. 在**流程管理**页面选择目标命名空间，然后单击**创建工作流**。
6. 在**创建工作流**面板，设置工作流的**名称**、**描述**、**应用ID**和**时间类型**（包括Cron和API），然后单击**确定**。



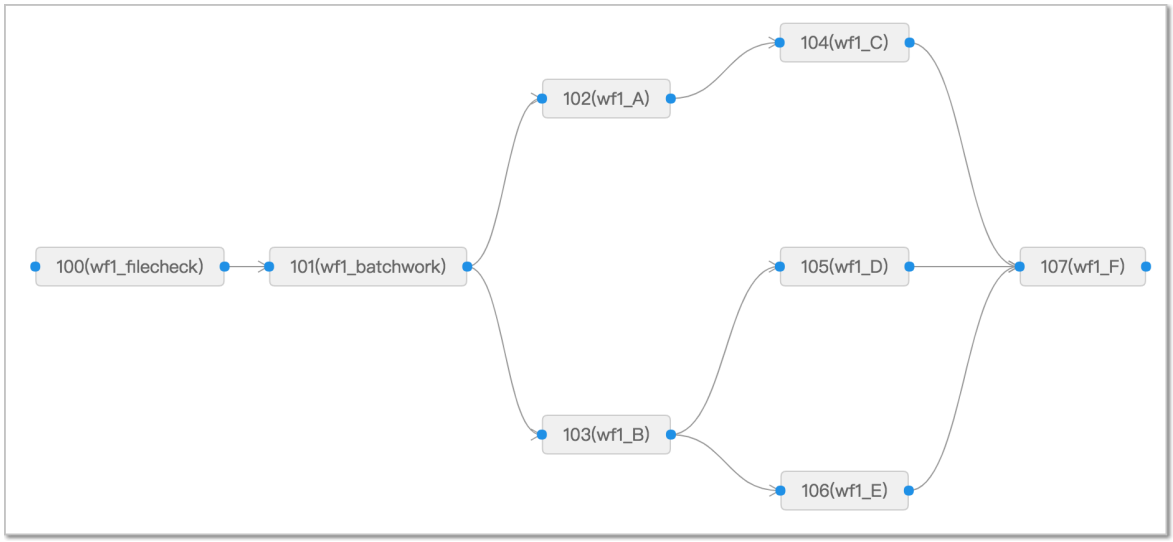
也可以单击**高级设置**，设置**时区**和**实例并发数**。

7. 在**工作流详情**页面，单击**创建任务**或**导入任务**，添加调度任务。
   - **创建任务**：和创建调度任务的步骤一致，请参见创建调度任务。
   - **导入任务**：将已创建的Job导入到工作流中。

   > **说明** 导入Job会弹出 `导入Job将会移除该Job的定时配置` 提示框，单击**确认**，则该Job不会独立调度，会跟随工作流的调度周期进行调度。

8. 调度任务添加完毕后，按住并拖动任务两侧的端点到其它任务的端点连接调度任务，形成依赖关系，组成完成的工作流。

   在工作流中，上下游的任务还可以实现数据传递。
   如果需要删除某两个任务间的依赖关系，选中它们之间的线条，单击Delete；右键某个调度任务，在快捷菜单中单击**删除**，即可删除该任务。
   一个工作流示意如下：



在该工作流中，101执行完，102和103会同时开始执行。104，105，106都执行成功，107才会开始执行。

9. 工作流配置完成后，单击**发布**。

   工作流创建完成后，返回**流程管理**页面，可以查看是否已经包含创建的流程及相关信息。

后续操作

工作流发布之后，就会变成启用状态并自动开始调度。如果不想马上启用，可以返回**流程管理**页面，在**操作**列单击**运行一次**测试下，没问题再手动启用。

您还可以对工作流进行编辑、查看操作记录和历史记录，以及删除、重刷任务等操作。

6.5.3. API 触发

您可以使用 API 触发任务调度。
API 触发的使用方式请参见调用方式。

6.5.4. 如何重刷数据

通过重刷数据功能，您可以重新触发一段时间区间内的实例，来重刷业务数据。

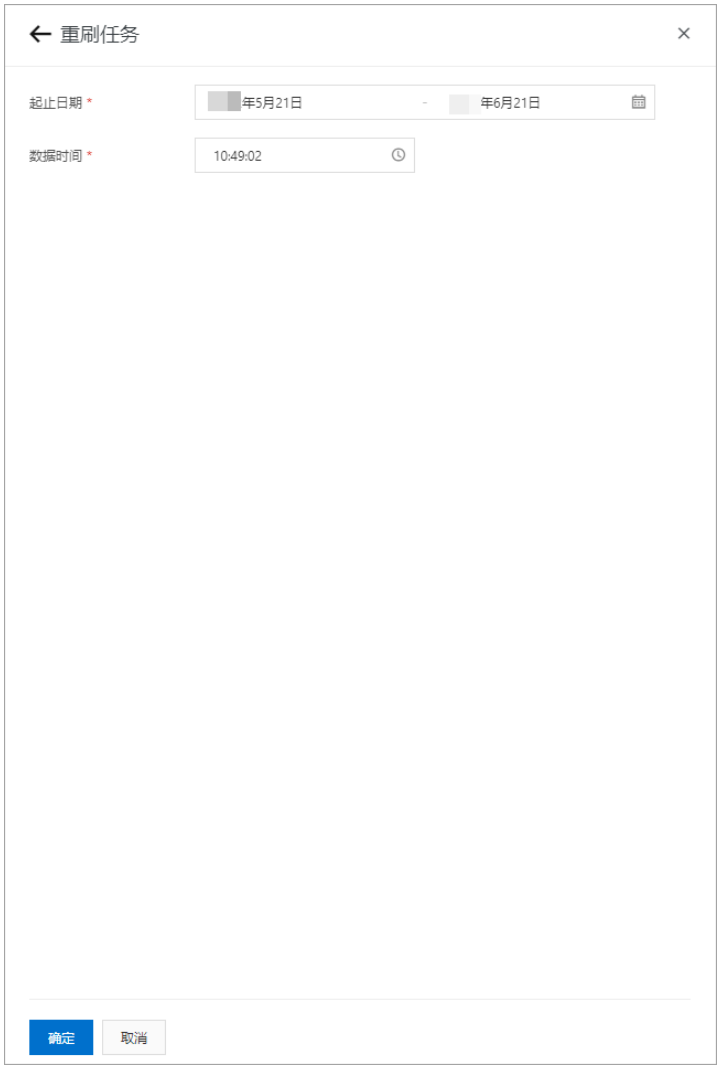
重刷调度任务

如果您的业务发生变更，如数据库增加一个字段或者上一个月数据有错误，需要把过去一段时间的任务重新执行一遍，可以重刷调度任务数据。

> **说明** 任务和工作流都支持重刷数据（只支持天级别的调度周期）。

如果您之前执行的某个调度任务的数据出现偏差或遗漏，您可以通过重新设置执行参数并执行某个调度任务属性、获取数据。

1. 在**任务管理**页面，单击目标任务**操作**列下的图标，然后单击**重刷任务**。
2. 在**重刷任务**面板，设置**起止日期**和**数据时间**，单击**确定**。
   - **起止日期**：指定重刷的日期区间。
   - **数据时间**：指定重刷日期区间内的重刷时间。



示例重刷配置如下：

- 当前时间为2019-01-01 10:00:00。
- 重刷任务的起止日期为2018-10-01~2018-10-07，默认从2018年10月1日00:00:00起，到2018年10月7日23:59:59结束。
- 数据时间为11:11:11。

则该任务会被重刷7次，生成7个实例。

序号	调度时间	数据时间
1	2019.1.1 10:00:00	2018.10.1 11:11:11
2	2019.1.1 10:00:00	2018.10.2 11:11:11
3	2019.1.1 10:00:00	2018.10.3 11:11:11
4	2019.1.1 10:00:00	2018.10.4 11:11:11
5	2019.1.1 10:00:00	2018.10.5 11:11:11
6	2019.1.1 10:00:00	2018.10.6 11:11:11
7	2019.1.1 10:00:00	2018.10.7 11:11:11

6.6. 任务类型

6.6.1. Java任务

Java调度任务可以在您的应用进程中执行，也可以通过上传JAR包来动态加载。

执行模式

Java任务类型支持单机、广播、并行计算、内存网格、网格计算和分片运行6种执行模式：

- *单机*：在同一个 `groupId` 下的机器随机挑一台执行。
- *广播*：同一个 `groupId` 下的所有机器同时执行。
- *并行计算*：支持子任务300以下，有子任务列表。
- *内存网格*：基于内存计算，子任务50,000以下，速度快。
- *网格计算*：基于文件计算，子任务1,000,000以下。
- *分片运行*：包括静态分片和动态分批，用于处理大数据业务需求。

*单机*和*广播*需要实现JavaProcessor；*并行计算*、*内存网格*、*网格计算*和*分片运行*需要实现MapJobProcessor。

Processor类路径，即实现类的全路径名，例如 `com.apache.armon.test.schedulerx.processor.MySimpleJob` ：

- 如果不上传JAR包，SchedulerX会去您的应用进程中的classpath下查找processor实现类，所以每次修改需要重新编译和发布。
- 如果上传了JAR包，每次会热加载JAR包和processor，不需要重新发布应用。

编程模型

Java任务支持两种编程模型：JavaProcessor和MapJobProcessor。

- JavaProcessor
  - （可选） `public void preProcess(JobContext context) throws Exception`
  - `public ProcessResult process(JobContext context) throws Exception`
  - （可选） `public void postProcess(JobContext context)`
  - （可选） `public void kill(JobContext context)`
- MapJobProcessor
  - `public ProcessResult process(JobContext context) throws Exception`
  - （可选） `public void postProcess(JobContext context)`
  - （可选） `public void kill(JobContext context)`
  - `public ProcessResult map(List<? extends Object> taskList, String taskName)`

ProcessResult

每个process需要返回ProcessResult，用来表示任务执行的状态、结果和错误信息。

- 任务运行成功： `return new ProcessResult(true)` 。
- 任务运行失败： `return new ProcessResult(false, ErrorMsg)` 或者直接抛异常。
- 任务运行成功并且返回结果： `return new ProcessResult(true, result)` 。 `result` 是一个字符串，不能大于1000字节。

HelloSchedulerx2.0任务示例

```
@Component
public class MyProcessor1 extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        //TODO
        System.out.println("Hello, schedulerx2.0!");
        return new ProcessResult(true);
    }
}
```

支持Kill功能的任务示例

```
@Component
public class MyProcessor2 extends JavaProcessor {
    private volatile boolean stop = false;
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        int N = 10000;
        while (!stop && N >= 0) {
            //TODO
            N--;
        }
        return new ProcessResult(true);
    }
    @Override
    public void kill(JobContext context) {
        stop = true;
    }
    @Override
    public void preProcess(JobContext context) {
        stop = false;  //如果是通过Spring启动，Bean是单例，需要通过preProcess把标记为复位
    }
}
```

通过Map模型批量处理任务示例

```
/**
 * 对一张单表进行分布式批量处理
 * 1. 根任务先查询一张表，获取minId，maxId
 * 2. 构造PageTask，通过map进行分发
 * 3. 下一级获取到如果是PageTask，则进行数据处理
 *
 */
@Component
public class ScanSingleTableJobProcessor extends MapJobProcessor {
    private static final int pageSize = 100;
    static class PageTask {
        private int startId;
        private int endId;
        public PageTask(int startId, int endId) {
             this.startId = startId;
             this.endId = endId;
        }
        public int getStartId() {
              return startId;
        }
        public int getEndId() {
              return endId;
        }
    }
    @Override
    public ProcessResult process(JobContext context) {
        String taskName = context.getTaskName();
        Object task = context.getTask();
        if (isRootTask(context)) {
            System.out.println("start root task");
            Pair<Integer, Integer> idPair = queryMinAndMaxId();
            int minId = idPair.getFirst();
            int maxId = idPair.getSecond();
            List<PageTask> taskList = Lists.newArrayList();
            int step = (int) ((maxId - minId) / pageSize); //计算分页数量
            for (int i = minId; i < maxId; i+=step) {
                taskList.add(new PageTask(i, (i+step > maxId ? maxId : i+step)));
            }
            return map(taskList, "Level1Dispatch"); //process调用map方法完成子任务分发
        } else if (taskName.equals("Level1Dispatch")) {
            PageTask record = (PageTask)task;
            long startId = record.getStartId();
            long endId = record.getEndId();
            //TODO
            return new ProcessResult(true);
        }
        return new ProcessResult(true);
    }
    @Override
    public void postProcess(JobContext context) {
        //TODO
        System.out.println("all tasks is finished.");
    }
    private Pair<Integer, Integer> queryMinAndMaxId() {
        //TODO select min(id),max(id) from xxx
        return null;
    }
}
```

6.6.2. 脚本任务

您可以在创建任务时直接编写Shell、Python和Go脚本以便创建脚本任务。

接入方式

脚本任务支持两种接入方式：

- 嵌在应用进程中
- 安装schedulerx2-agent

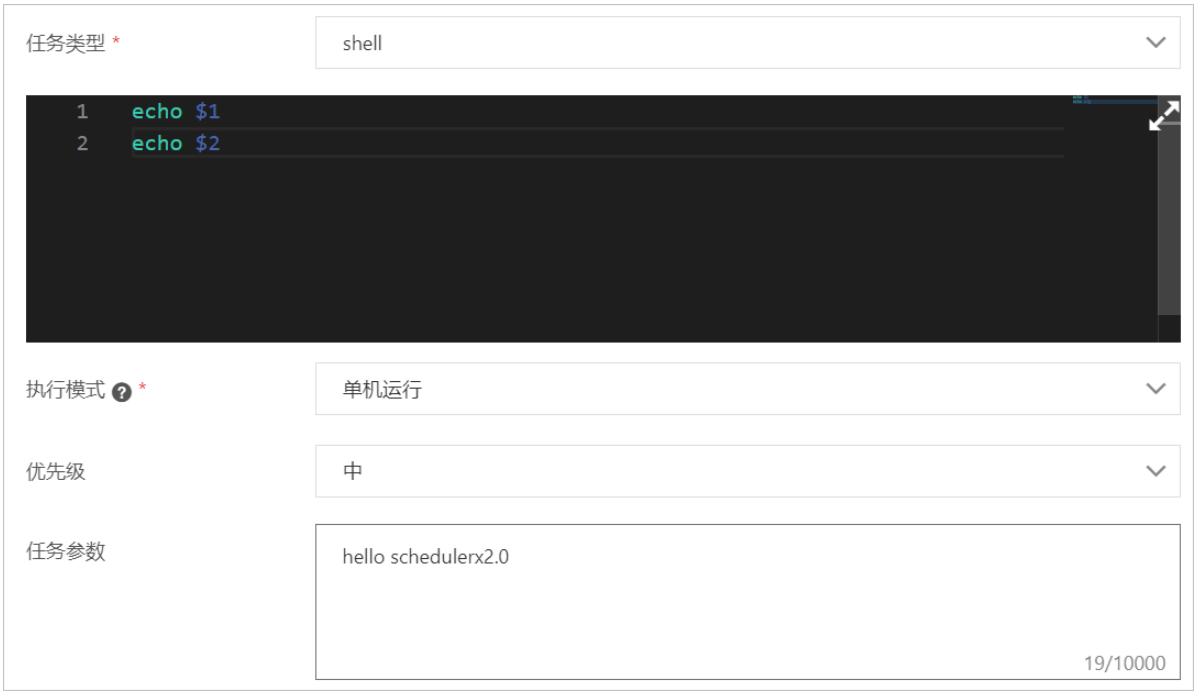
创建方式

脚本任务在创建调度任务时编写脚本即可，详情请参见创建调度任务。

编写脚本的方式包括创建无参数的Shell任务、带参数的Shell任务、Python任务和Go任务：

- 无参数的Shell任务示例。

- 带参数的Shell任务示例。



- Python任务示例。

- Go任务示例。

6.6.3. HTTP任务（Serverless）

SchedulerX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。

使用限制

- 目前只支持GET、POST，后续根据用户需求陆续开通其他方法。
- HTTP请求返回结果必须是JSON格式，服务需要解析指定key比较本次请求是否成功。
- 不支持秒级任务，支持到分钟级别。
- 请求URL需要有公网权限，如果是 `IP:port` ，需要机器开通公网权限。

GET

使用HTTP的GET方法，需要在客户端中添加配置，然后在控制台中创建任务。

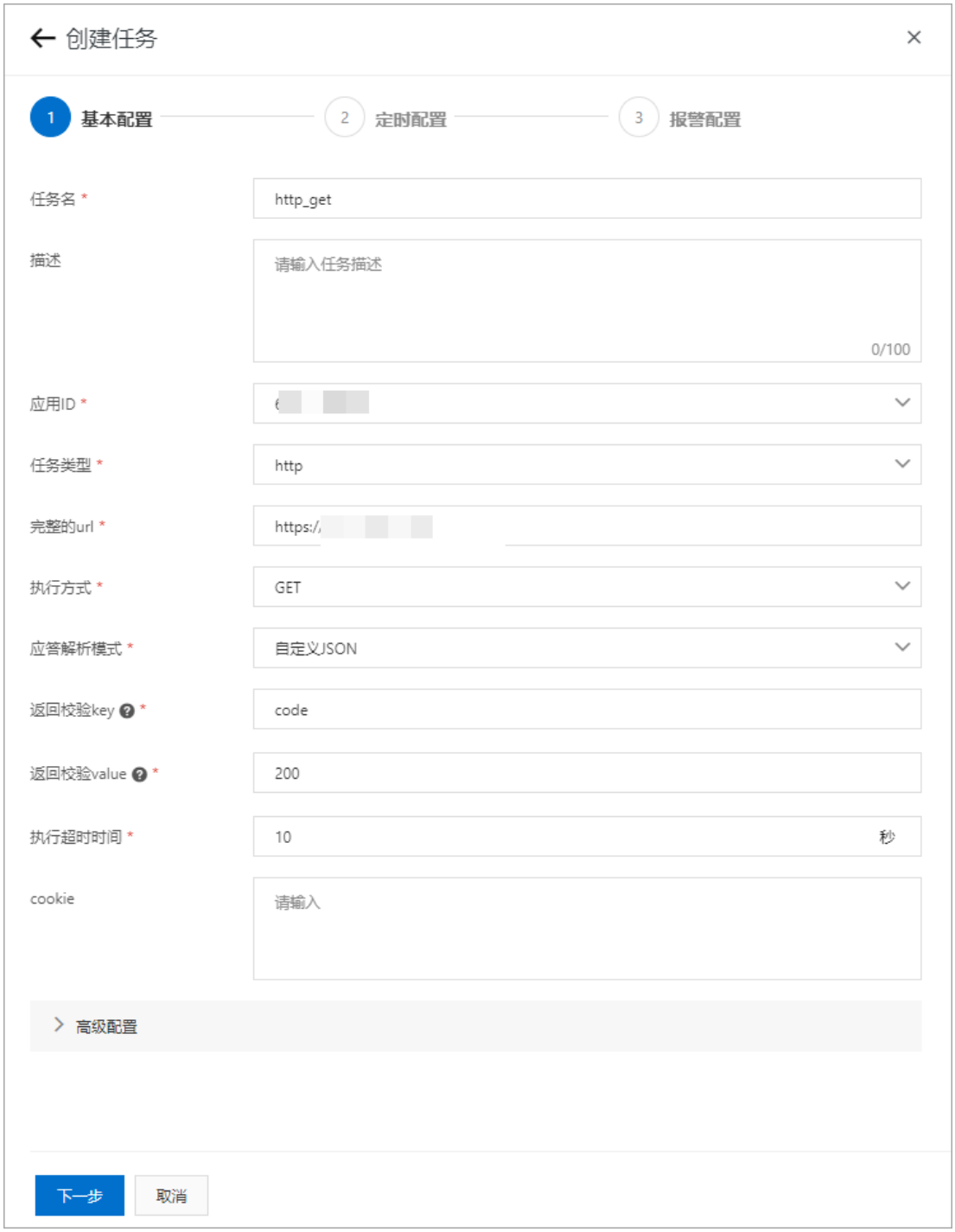
1. 在客户端中添加配置GET方法配置。

   客户端接入SchedulerX的详细步骤，请参见Spring Boot应用接入SchedulerX。此处仅介绍GET方法的配置。

```
@GET
@Path("hi")
@Produces(MediaType.APPLICATION_JSON)
public RestResult hi(@QueryParam("user") String user) {
    TestVo vo = new TestVo();
    vo.setName(user);
    RestResult result = new RestResult();
    result.setCode(200);
    result.setData(vo);
    return result;
}
```

2. 在控制台创建HTTP任务。

   创建调度任务，请参见创建调度任务。此处仅介绍HTTP任务GET方法相关配置。



Serverless HTTP任务参数说明：

参数	说明
完整的URL	需要填写完整URL，包括 `http://` 。

参数	说明
返回校验key和返回校验value	服务端默认HTTP请求结果为JSON格式，根据填写的key和value校验结果是否成功。 `{ code: 200, data: "true", message: "", requestId: "4466550687919236141033812329 71", success: true }` 上面的示例代码可以校验key为success，校验`value:true`或者校验code是否为200。
执行超时时间（秒）	最大30秒，超过会报错。
cookie	格式例如`key1=val1;key2=val2`，多个值用半角分号（;）隔开，最大长度为300字节。

3. 任务创建成功后，在**任务管理**页面的**操作**列单击**运行一次**。
出现以下结果，说明任务执行成功。

POST

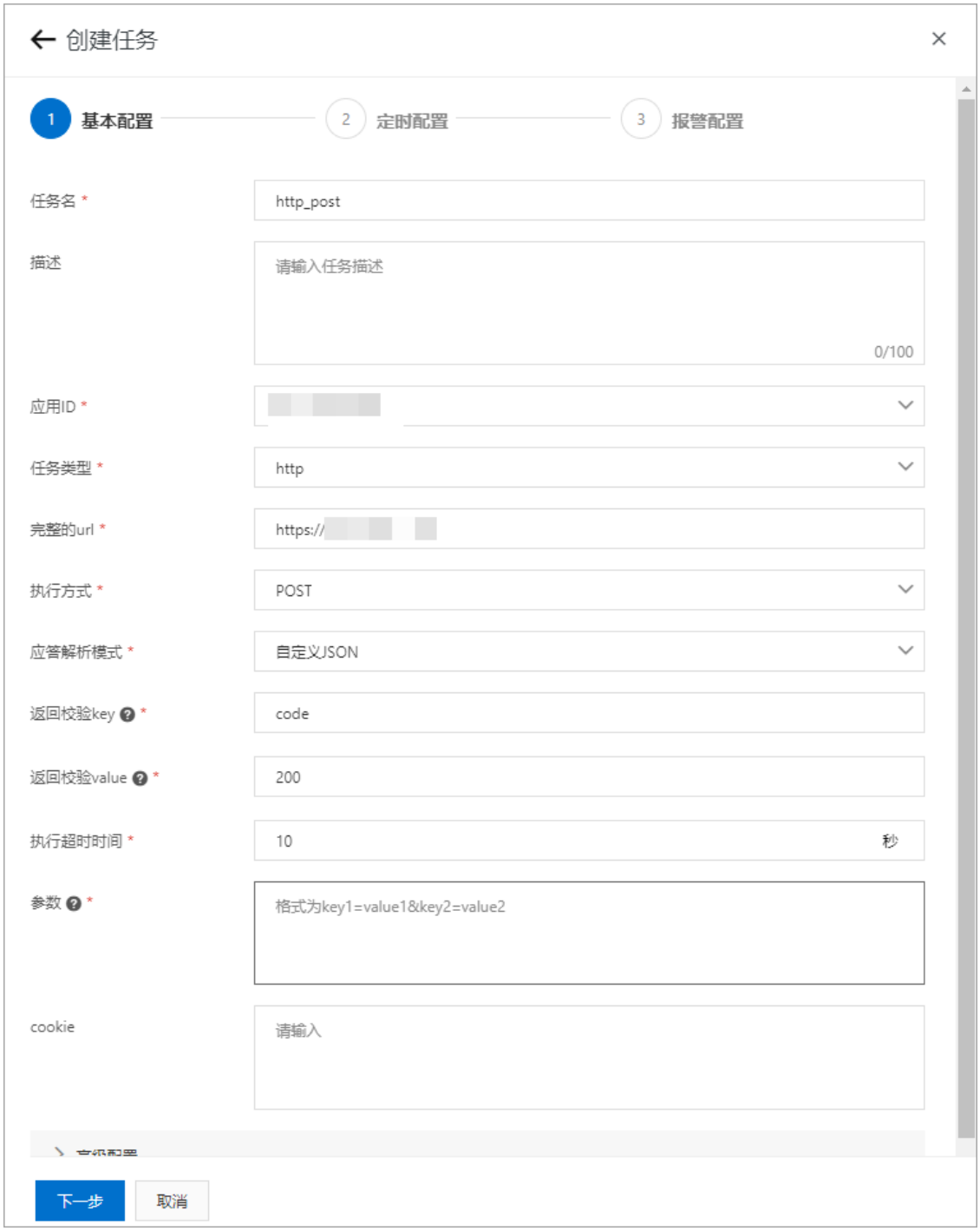
使用HTTP的POST方法，需要在客户端中添加配置，然后在控制台中创建任务。

1. 在客户端中添加配置POST方法配置。
客户端接入SchedulerX的详细步骤，请参见Spring Boot应用接入SchedulerX。此处仅介绍POST方法的配置。

```
import com.alibaba.schedulerx.common.constants.CommonConstants;
@POST
@Path("createUser")
@Produces(MediaType.APPLICATION_JSON)
public RestResult createUser(@FormParam("userId") String userId,
        @FormParam("userName") String userName) {
    TestVo vo = new TestVo();
    System.out.println("userId=" + userId + ", userName=" + userName);
    vo.setName(userName);
    RestResult result = new RestResult();
    result.setCode(200);
    result.setData(vo);
    return result;
}
```

2. 在控制台创建HTTP任务。

创建调度任务，请参见创建调度任务。此处仅介绍HTTP任务POST方法相关配置。



Serverless HTTP任务参数说明：

参数	说明
完整的URL	需要填写完整URL，包括 `http://` 。

参数	说明
返回校验key和返回校验value	服务端默认HTTP请求结果为JSON格式，根据填写的key和value校验结果是否成功。 `{ code: 200, data: "true", message: "", requestId: "4466550687919236141033812329 71", success: true }` 上面的示例代码可以校验key为success，校验`value:true`或者校验code是否为200。
执行超时时间（秒）	最大30秒，超过会报错。
参数	POST表单参数，格式例如`key1=val1;key2=val2`。

如何获取任务基本信息

HTTP任务，会将任务基本信息打入header中，如果想获取任务的基本信息，可以在客户端的*pom.xml*中增加以下依赖。

```
<dependency>
    <groupId>com.aliyun.schedulerx</groupId>
    <artifactId>schedulerx2-common</artifactId>
    <version>1.1.5-SNAPSHOT</version>
</dependency>
```

以GET方法为例，通过以下方式获取任务基本信息。

```
import com.alibaba.schedulerx.common.constants.CommonConstants;
@GET
@Path("hi")
@Produces(MediaType.APPLICATION_JSON)
public RestResult hi(@QueryParam("user") String user,
        @HeaderParam(CommonConstants.JOB_ID_HEADER) String jobId,
        @HeaderParam(CommonConstants.JOB_NAME_HEADER) String jobName) {
    TestVo vo = new TestVo();
    vo.setName("armon");
    //JobName可能是中文，需要URLDecode。
    String decodedJobName = URLDecoder.decode(jobName, "utf-8");
    System.out.println("user=" + user + ", jobId=" + jobId + ", jobName=" + decodedJobName);
    RestResult result = new RestResult();
    result.setCode(200);
    result.setData(vo);
    return result;
}
```

可以获取的任务基本信息如下：

CommonConstants常量	key	value描述
JOB_ID_HEADER	schedulerx-jobId	任务ID

CommonConstants常量	key	value描述
JOB_NAME_HEADER	schedulerx-jobName	任务名，需要引文，因为header不支持中文。
SCHEDULE_TIMESTAMP_HEADER	schedulerx-scheduleTimestamp	调度时间的时间戳
DATA_TIMESTAMP_HEADER	schedulerx-dataTimestamp	数据时间的时间戳
GROUP_ID_HEADER	schedulerx-groupId	应用ID
USER_HEADER	schedulerx-user	用户名
MAX_ATTEMPT_HEADER	schedulerx-maxAttempt	实例最大重试次数
ATTEMPT_HEADER	schedulerx-attempt	实例当前重试次数
JOB_PARAMETERS_HEADER	schedulerx-jobParameters	任务参数
INSTANCE_PARAMETERS_HEADER	schedulerx-instanceParameters	任务实例参数，需要API触发

结果验证

HTTP任务执行结果在执行列表页可以进行查询，成功结果可参见GET中的结果。

如果失败可以单击**详情**查看具体失败原因，下面列举典型的失败结果。

- 返回值和期望不相同

- 执行超时

报警

HTTP任务支持错误报警，针对上述超时以及返回值不符合预期用户可以在创建任务时设置报警信息，接收相应的报警信息。

6.6.4. DataWorks任务

SchedulerX可支持定时调度DataWorks任务，并将DataWorks任务与其他任务在SchedulerX上进行混合依赖编排，完成相应的定期任务数据处理。

前提条件

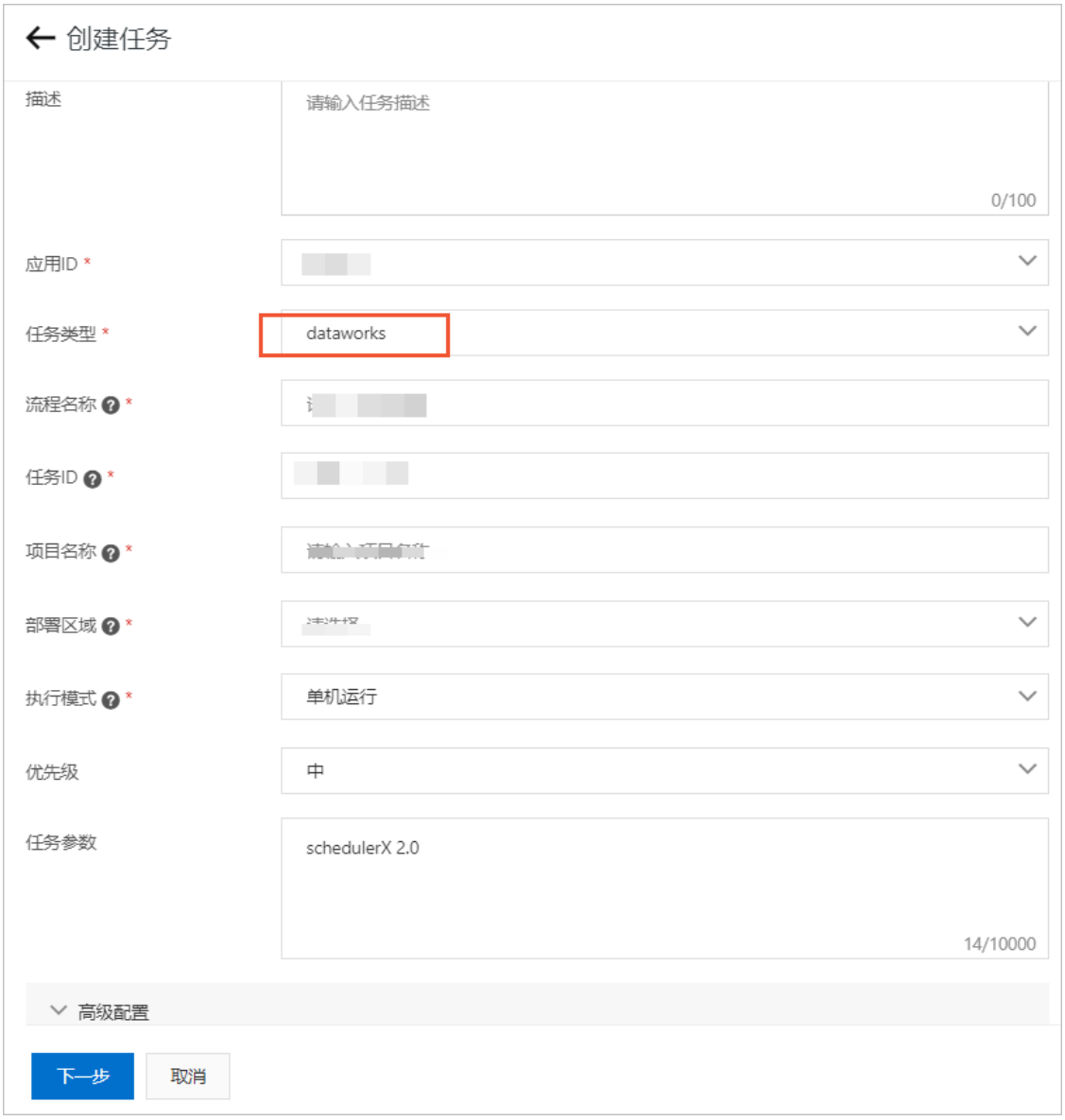
- SchedulerX客户端升级至1.3.4及以上版本。
- 接入客户端，将具备访问DataWorks（需开通企业版）权限的用户AccessKey ID和AccessKey Secret写入*agent.properties*文件，请参见Agent接入（非Java SDK方式）。

```
# DataWorks访问的账户信息配置
spring.SchedulerX2.aliyunAccessKey=阿里云账号AccessKey ID
spring.SchedulerX2.aliyunSecretKey=阿里云账号AccessKey Secret
```

任务创建

1. 在DataWorks控制台上执行如下操作：
   i. 创建手动业务流程，请参见创建手动业务流程。
   ii. 创建节点（不需配置依赖关系），请参见创建节点并配置依赖关系。
   iii. 提交业务流程，请参见提交业务流程。
2. 在SchedulerX控制台上执行如下操作：

i. 创建相应工作流并绑定DataWorks任务节点，请参见创建工作流。



ii. 任务添加完毕后，按住并拖动任务两侧的端点到其他任务的端点连接调度任务，形成依赖关系，组成完整的工作流。

iii. 对创建的业务流程设置定时触发，具体操作，请参见Cron。

执行结果

创建的定时任务触发后，可在流程实例列表中查看流程执行状态以及各个节点状态，鼠标右键单击节点可查询对应节点的执行结果的详细信息。

后续步骤

- 在SchedulerX任务实例列表，您可查看对应任务节点的执行详情，进行任务停止、重跑等操作。
- 在DataWorks的运维中心，您可查询本次调度的执行实例信息。

6.6.5. XxlJob任务

SchedulerX 2.0兼容XXL-JOB任务接口，支持 `@XxlJob` 新注解和 `@JobHandler` 老注解方式，您不需要修改代码，即可以将XXL-JOB任务在SchedulerX 2.0平台上进行调度。

背景信息

XXL-JOB是一个开箱即用的轻量级分布式任务调度系统，其核心设计目标是开发迅速、学习简单、轻量级、易扩展，在开源社区广泛流行，已在多家公司投入使用。XXL-JOB开源协议采用的是GPL，因此云厂商无法直接商业化托管该产品，各大中小企业需要自建，增加了学习成本、机器成本、人工运维成本。阿里巴巴商业化任务调度平台SchedulerX 2.0兼容XXL-JOB任务接口，您不需要修改一行代码，即可以将XXL-JOB任务在SchedulerX 2.0平台上托管。

采用托管的XXL-JOB有以下优势：

- **免运维、低成本**
  自建XXL-JOB最少需要2个服务器和1个数据库支撑，而使用托管的XXL-JOB可以省去这些机器成本和人力运维成本。
- **海量任务、精准调度**
  开源XXL-JOB基于竞争数据库锁保证只有一个节点执行任务，对于数据库有压力。据统计，当任务超过1万，都是分钟级别的任务时，就会有比较明显的调度延时，如果是秒级别任务，延时就更加明显。SchedulerX 2.0采用分布式架构，不同的server调度不同的任务，且无锁竞争，真正实现可以水平扩展，可以支持百万级别任务调度。
  SchedulerX 2.0针对秒级别任务低延时的特性，采用了专门的架构，占用资源极低，可以作为实时业务的秒级别调度场景。另外，SchedulerX 2.0还支持一次性任务，可以指定未来某个时刻执行一次任务，执行完任务自动销毁，可以作为定时通知、订单定时关闭等场景。
- **高级特性**
  - 工作流：通过可视化的工作流进行任务编排。
  - 限流：可抢占的任务优先级队列。
  - 资源隔离：支持命名空间和应用级别资源隔离，支持多租户权限管理。
- **高可用**
  SchedulerX 2.0采用高可用架构，任务多备份机制，经历过阿里集团多年双十一、容灾演练等场景的考验，可以做到整个集群挂掉任意2个节点或者任意一个机房断电，任务调度都不会收到影响。
- **商业化报警运维**
  SchedulerX 2.0除了兼容XXL-JOB的邮件报警和基本运维操作，还提供了商业化报警和运维功能：
  - 报警：通过钉群、短信、电话等发送通知。
  - 运维：支持原地重跑、重刷数据、标记成功、查看堆栈、停止任务等操作。

与开源XXL-JOB的区别

功能	开源XXL-JOB	SchedulerX为底座的XXL-JOB任务
Bean模式	不兼容新版本 `@XxlJob` 注解和老版本 `@JobHandler` 注解	同时兼容 `@XxlJob` 注解和 `@JobHandler` 注解
GLUE（Java）	支持	不支持
Shell	支持	支持
Nodejs	支持	支持
HTTP	不支持	支持
单机	支持	支持
分片广播	支持	支持
路由策略	第一个、最后一个、轮询、随机等	轮询
定时	cron	cron、fixed_rate、fixed_delay、one_time
工作流	不支持	支持

功能	开源XXL-JOB	SchedulerX为底座的XXL-JOB任务
运维操作	运行一次	运行一次、原地重跑、重刷数据、标记成功、停止运行
报警	邮件	邮件、钉钉群、短信、电话

接入配置

1. 应用的*pom.xml*文件做如下变更：将 `com.xuxueli:XXL-JOB-core` 的依赖去除，增加SchedulerX客户端的依赖和 `com.aliyun:schedulerx2-plugin-xxljob` 插件。
   因为XXL-JOB在2.3.x版本重构了接口，请根据XXL-JOB的版本选择不同的接入方式，以 `schedulerx2-spring-boot-starter` 应用为例，配置内容如下：
   - 2.3.x版本接入

     ```
     <!-- 注释xxl-job-core -->
     <!--
     <dependency>
         <groupId>com.xuxueli</groupId>
         <artifactId>xxl-job-core</artifactId>
         <version>${project.parent.version}</version>
     </dependency>
     -->
     <dependency>
         <groupId>com.aliyun.schedulerx</groupId>
         <artifactId>schedulerx2-spring-boot-starter</artifactId>
         <version>1.4.0</version>
     </dependency>
     <dependency>
         <groupId>com.aliyun.schedulerx</groupId>
         <artifactId>schedulerx2-plugin-xxljob</artifactId>
         <version>2.3.0</version>
     </dependency>
     ```

   - 2.3.x以下版本接入

     ```
     <!-- 注释xxl-job-core -->
     <!--
     <dependency>
         <groupId>com.xuxueli</groupId>
         <artifactId>xxl-job-core</artifactId>
         <version>${project.parent.version}</version>
     </dependency>
     -->
     <dependency>
         <groupId>com.aliyun.schedulerx</groupId>
         <artifactId>schedulerx2-spring-boot-starter</artifactId>
         <version>1.4.0</version>
     </dependency>
     <dependency>
         <groupId>com.aliyun.schedulerx</groupId>
         <artifactId>schedulerx2-plugin-xxljob</artifactId>
         <version>1.3.4.1</version>
     </dependency>
     ```

2. *agent.properties*文件添加如下配置，具体操作，请参见Agent接入（非Java SDK方式）。

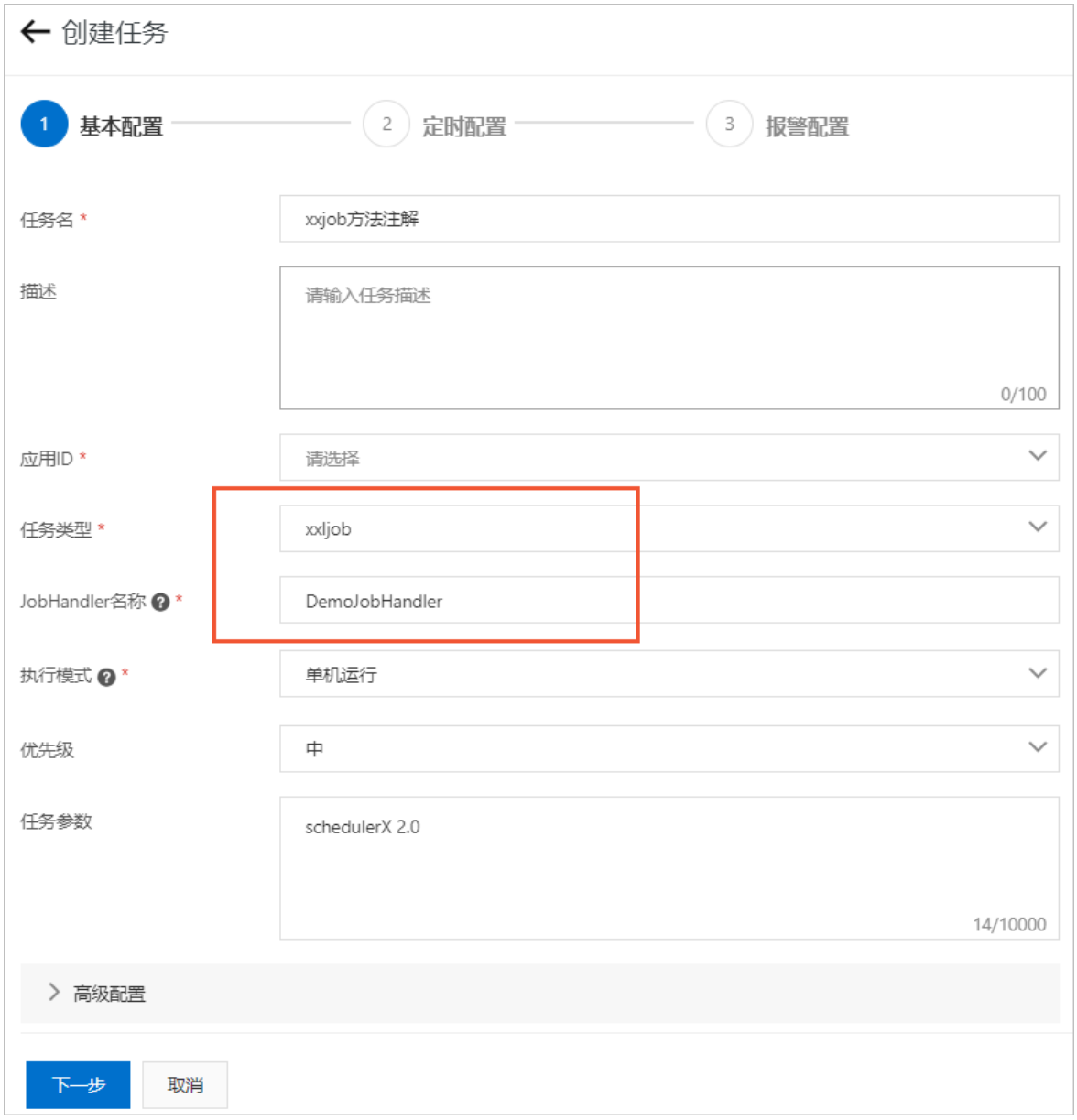
```
spring.schedulerx2.endpoint=192.xx.xx.xx
spring.schedulerx2.namespace=20e90ffc****
spring.schedulerx2.groupId=679xxx
spring.schedulerx2.appKey=71BCC0Exxx
```

使用Xxljob方法创建任务

参考开源 `XXL-JOB-executor-sample-springboot` 工程，创建方法任务，示例代码如下：

```
/**
* 简单任务示例（Bean模式）
*/
   @XxlJob("demoJobHandler")
    public ReturnT<String> demoJobHandler(String param) throws Exception {
        System.out.println("XXL-JOB, " + param);
        return ReturnT.SUCCESS;
    }
```

1. 在左侧导航栏单击**任务管理**。
2. 在应用列表页面顶部菜单栏选择**地域**，在页面中选择目标**微服务空间**，然后单击**创建任务**。参数配置，请参见创建调度任务。



3. 在**任务管理**页面，单击目标任务**操作**列下的**运行一次**。
4. 在**任务管理**页面，单击目标任务**操作**列下的图标，单击**历史记录**，查看任务参数。

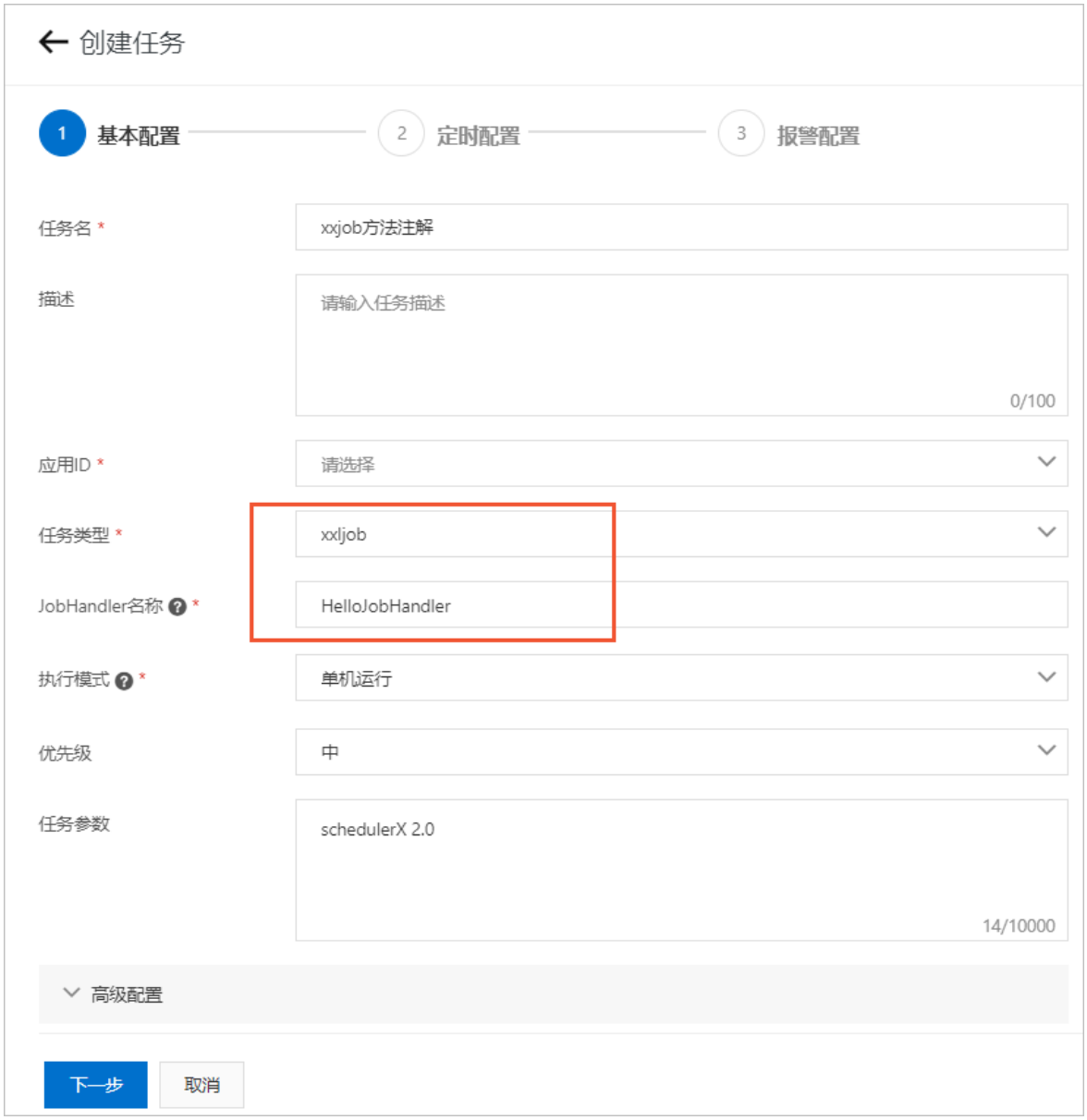
使用JobHandler方法创建任务

示例代码如下：

```
@JobHandler(value="HelloJobHandler")
@Componentpublic class HelloJobHandler extends IJobHandler {
    @Override
    public ReturnT<String> execute(String param) throws Exception {
        System.out.println("HelloJobHandler: " + param);
        return SUCCESS;
    }
}
```

1. 在左侧导航栏单击**任务管理**。

2. 在应用列表页面顶部菜单栏选择**地域**，在页面中选择目标**微服务空间**，然后单击**创建任务**。参数配置，请参见创建调度任务。



3. 在**任务管理**页面，单击目标任务**操作**列下的**运行一次**。

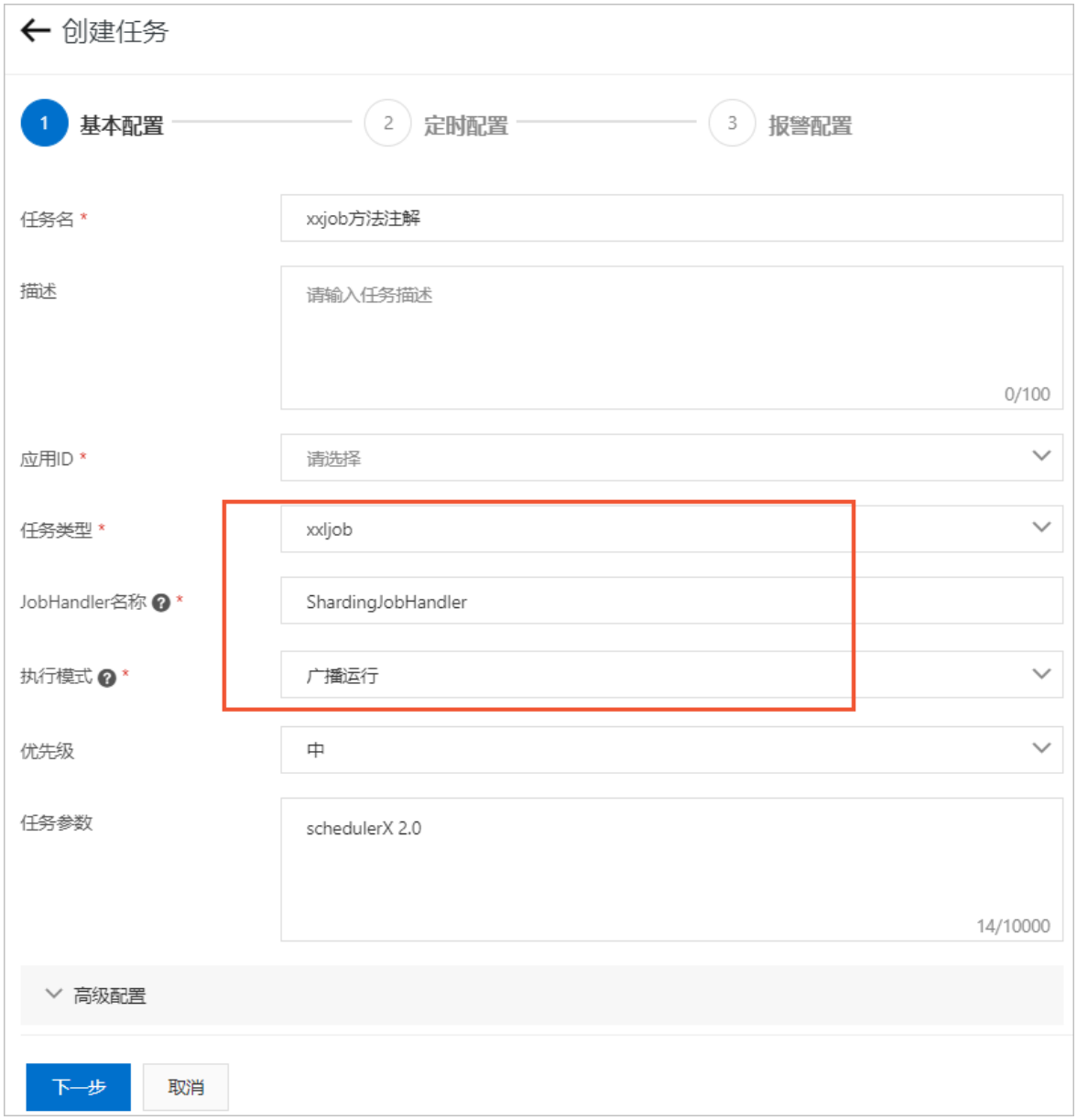
4. 在**任务管理**页面，单击目标任务**操作**列下的图标，单击**历史记录**，查看任务参数。

使用分片广播的方式创建任务

以XxlJob方法注解为例，示例代码如下：

```
/**
* 分片广播任务
*/
    @XxlJob("shardingJobHandler")
    public ReturnT<String> shardingJobHandler(String param) throws Exception {
        // 分片参数
        int shardIndex = XxlJobContext.getXxlJobContext().getShardIndex();
        int shardTotal = XxlJobContext.getXxlJobContext().getShardTotal();
        XxlJobLogger.log("分片参数：当前分片序号 = {}, 总分片数 = {}", shardIndex, shardTotal);
        System.out.println("分片参数：当前分片序号 ="+ shardIndex + ", 总分片数 = " + shardTotal);
        // 业务逻辑
        for (int i = 0; i < shardTotal; i++) {
            if (i == shardIndex) {
                System.out.println("第 " + i + " 片，命中分片开始处理");
                XxlJobLogger.log("第 {} 片，命中分片开始处理", i);
            } else {
                XxlJobLogger.log("第 {} 片，忽略", i);
            }
        }
        return ReturnT.SUCCESS;
    }
```

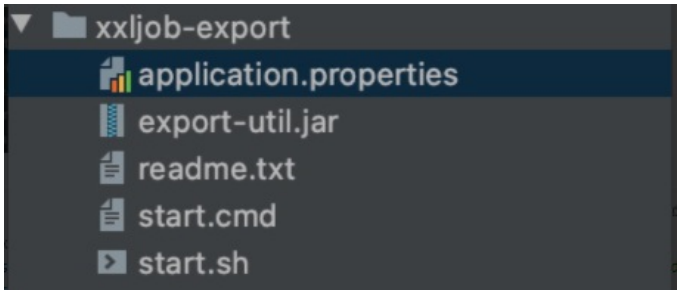
1. 在左侧导航栏单击**任务管理**。
2. 在应用列表页面顶部菜单栏选择**地域**，在页面中选择目标**微服务空间**，然后单击**创建任务**。参数配置，请参见创建调度任务。



3. 在**任务管理**页面，单击目标任务**操作**列下的**运行一次**。
4. 在**任务管理**页面，单击目标任务**操作**列下的图标，单击**历史记录**，查看任务参数。

XXL-JOB任务批量迁移

- XXL-JOB导出f任务配置
  i. 下载导出工具。
  ii. 解压工具包，在 `application.properties` 文件中进行导出配置。



配置参数如下：

```
### xxl-job, datasource
datasource.url=jdbc:mysql://127.0.0.1:3306/xxl_job?useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&serverTimezone=Asia/Shanghai
datasource.username=root
datasource.password=123456
datasource.driver-class-name=com.mysql.cj.jdbc.Driver
### 配置对应要导出的app-name（该步骤可选，不设置的情况下导出所有任务分组）
#xxl-job.app-name=xxl-job-executor-sample
```

iii. 执行 `./start.sh` 命令运行导出操作。

```
yaohui( ; xxljob-export % ./start.sh
16:31:35.866 logback [main] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
16:31:36.411 logback [main] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
<xxl-job-executor-sample> start export job...
<xxl-job-executor-sample> finish export job size<4>
<xxxx> start export job...
<xxxx> finish export job size<0>
export finished.
```

iv. 执行完成后，在当前目录下获得 `xxljob_*******.json` 的任务配置文件（仅Bean模式的任务配置信息）。

- XXL-JOB任务导入SchedulerX 2.0
  i. 登录SchedulerX 2.0控制台。
  ii. 在左侧导航栏单击**任务管理**。
  iii. 在**任务管理**页面上方单击**导入任务**。
  iv. 选择待导入的任务配置文件后单击**导入**，即可完成XXL-JOB任务配置信息和SchedulerX任务配置信息的同步。

6.7. 分布式编程模型

6.7.1. 单机

单机执行表示一个任务实例只会随机触发到一台Worker上，支持所有的任务类型。

6.7.2. 广播

广播执行表示一个任务实例会广播到该分组所有Worker上执行，当所有Worker都执行完成，该任务才算完成。任意一台Worker执行失败，都算该任务失败。

应用场景

- 批量运维
  - 定时广播所有机器运行某个脚本。
  - 定时广播所有机器清理缓存。
  - 动态拉起每台机器的某个服务，最后由一台机器回收结果修改数据库。
- 数据聚合

- 使用JavaProcessor，preProcess的时候初始化Redis缓存或者数据库。
- 每台机器执行process的时候，根据自己业务返回result。
- postProcess的时候，获取所有机器的执行结果做汇总，更新缓存或者数据库。

任务类型

任务类型可以选择多种，例如脚本或者Java任务。如果选择Java，还支持preProcess和postProcess高级特性。

使用Java任务需要继承JavaProcessor（1.0.7及以上版本），接口如下：

- `public ProcessResult process(JobContext context) throws Exception`
- （可选）`public void preProcess(JobContext context)`
- （可选）`public ProcessResult postProcess(JobContext context)`

preProcess会在所有机器执行process之前执行，且只会执行一次。

postProcess会在所有机器执行process且都成功执行之后执行一次，可以返回结果，作为工作流数据传输。

Demo示例

```
@Component
public class TestBroadcastJob extends JavaProcessor {
    /**
     * 只有一台机器会执行
     */
    @Override
    public void preProcess(JobContext context) {
        System.out.println("TestBroadcastJob.preProcess");
    }
    /**
     * 所有机器会执行
     */
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        int value = new Random().nextInt(10);
        System.out.println("分片总数=" + context.getShardingNum() + ", 分片号=" + context.getSharding
Id() + ", "
                + "taskId=" + context.getTaskId() + ", value=" + value);
        return new ProcessResult(true, String.valueOf(value));
    }
    /**
     * 只有一台机器会执行
     */
    @Override
    public ProcessResult postProcess(JobContext context) {
        System.out.println("TestBroadcastJob.postProcess");
        Map<Long, String> allTaskResults = context.getTaskResults();
        Map<Long, TaskStatus> allTaskStatuses = context.getTaskStatuses();
        int num = 0;
        for (Entry<Long, String> entry : allTaskResults.entrySet()) {
            System.out.println(entry.getKey() + ":" + entry.getValue());
            if (allTaskStatuses.get(entry.getKey()).equals(TaskStatus.SUCCESS)) {
                num += Integer.valueOf(entry.getValue());
            }
        }
        System.out.println("TestBroadcastJob.postProcess(), num=" + num);
        return new ProcessResult(true, String.valueOf(num));
    }
}
```

6.7.3. Map模型

基于MapJobProcessor，调用Map方法，即可实现大数据分布式跑批的能力。

注意事项

- SchedulerX不保证子任务一定执行一次，在特殊条件下会failover，可能会导致子任务重复执行，需要业务方自己实现幂等。
- SchedulerX使用的是Hessian序列化框架，目前不支持LocalDateTime和BigDecimal。子任务中如果有如上两个数据结构，请替换其他的数据结构（特别是BigDecimal，序列化不会报错，反序列化会变成0）。

接口

接口	解释	是否必选
`public ProcessResult process(JobContext context) throws Exception;`	每个子任务执行业务的入口，需要从context里获取taskName，自己判断是哪个子任务，进行相应的逻辑处理。执行完成后，需要返回ProcessResult。ProcessResult接口请参见ProcessResult。	是
`public void postProcess(JobContext context);`	无	否
`public void kill(JobContext context);`	前端kill任务会触发该方法，需要用户自己实现如何中断业务。	否
`public ProcessResult map(List<? extends Object> taskList, String taskName)`	执行map方法可以把一批子任务分布式到多台机器上执行，可以map多次。如果taskList是空，返回失败。执行完成后，需要返回ProcessResult。ProcessResult接口请参见ProcessResult。	是

执行方式

- 并行计算：最多支持300任务，有子任务列表。

  > **注意** 秒级别任务不要选择并行计算。

- 内存网格：基于内存计算，最多支持50,000以下子任务，速度快。
- 网格计算：基于文件计算，最多支持1,000,000子任务。

高级配置

任务管理高级配置参数说明如下：

参数	适用的执行模式	解释	默认值
实例失败重试次数	通用	任务运行失败自动重试的次数。	0
实例失败重试间隔	通用	每次失败重试的间隔。单位：秒。	30

参数	适用的执行模式	解释	默认值
实例并发数	通用	同一个Job同一时间运行的实例个数。1表示不允许重复执行。	1
子任务单机并发数	• 并行计算 • 内存网格 • 网格计算	分布式模型，单台机器并发消费子任务的个数。	5
子任务失败重试次数	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的次数。	0
子任务失败重试间隔	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的间隔。单位：秒。	0
子任务分发方式	• 并行计算 • 内存网格 • 网格计算	• **推模型**：每台机器平均分配子任务。 • **拉模型**： 每台机器主动拉取子任务，没有木桶效应。拉取过程中，所有子任务会缓存在Master节点，对内存有压力，建议子任务数不超过10,000。	推模型
子任务单次拉取数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点每次向Master节点拉取多少个子任务。	5
子任务队列容量（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点缓存子任务的队列大小。	10
子任务全局并发数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	分布式拉模型支持全局子任务并发数，可以进行限流。	1,000

发送50条消息的Demo示例（适用于Map模型）

```
@Component
public class TestMapJobProcessor extends MapJobProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        int dispatchNum = 50;
        if (isRootTask(context)) {
            System.out.println("start root task");
            List<String> msgList = Lists.newArrayList();
            for (int i = 0; i <= dispatchNum; i++) {
                msgList.add("msg_" + i);
            }
            return map(msgList, "Level1Dispatch");
        } else if (taskName.equals("Level1Dispatch")) {
            String task = (String)context.getTask();
            System.out.println(task);
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
    }
}
```

处理单表数据的Demo示例（适用于Map或MapReduce模型）

```
@Component
public class ScanSingleTableJobProcessor extends MapJobProcessor {
    @Service
    private XXXService xxxService;
    private final int PAGE_SIZE = 500;
    static class PageTask {
        private long startId;
        private long endId;
        public PageTask(long startId, long endId) {
            this.startId = startId;
            this.endId = endId;
        }
        public long getStartId() {
            return startId;
        }
        public long getEndId() {
            return endId;
        }
    }
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String tableName = context.getJobParameters(); //多个Job后端代码可以一致，通过控制台配置Job参数表示表名。
        String taskName = context.getTaskName();
        Object task = context.getTask();
        if (isRootTask(context)) {
            Pair<Long, Long> idPair = queryMinAndMaxId(tableName);
            long minId = idPair.getFirst();
            long maxId = idPair.getSecond();
            List<PageTask> tasks = Lists.newArrayList();
            int step = (int) ((maxId - minId) / PAGE_SIZE); //计算分页数量
            for (long i = minId; i < maxId; i+=step) {
                tasks.add(new PageTask(i, (i+step > maxId ? maxId : i+step)));
            }
            return map(tasks, "PageTask");
        } else if (taskName.equals("PageTask")) {
            PageTask pageTask = (PageTask)task;
            long startId = pageTask.getStartId();
            long endId = pageTask.getEndId();
            List<Record> records = queryRecord(tableName, startId, endId);
            //TODO handle records
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
    }
    private Pair<Long, Long> queryMinAndMaxId(String tableName) {
        //TODO select min(id),max(id) from [tableName]
        return new Pair<Long, Long>(1L, 10000L);
    }
    private List<Record> queryRecord(String tableName, long startId, long endId) {
        List<Record> records = Lists.newArrayList();
        //TODO select * from [tableName] where id>=[startId] and id<[endId]
        return records;
    }
}
```

处理分库分表数据的Demo示例（适用于Map或MapReduce模型）

```
@Component
public class ScanShardingTableJobProcessor extends MapJobProcessor {
    @Service
    private XXXService xxxService;
    private final int PAGE_SIZE = 500;
    static class PageTask {
        private String tableName;
        private long startId;
        private long endId;
        public PageTask(String tableName, long startId, long endId) {
            this.tableName = tableName;
            this.startId = startId;
            this.endId = endId;
        }
        public String getTableName() {
            return tableName;
        }
        public long getStartId() {
            return startId;
        }
        public long getEndId() {
            return endId;
        }
    }
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        Object task = context.getTask();
        if (isRootTask(context)) {
            //先分库
            List<String> dbList = getDbList();
            return map(dbList, "DbTask");
        } else if (taskName.equals("DbTask")) {
            //根据分库去分表
            String dbName = (String)task;
            List<String> tableList = getTableList(dbName);
            return map(tableList, "TableTask");
        } else if (taskName.equals("TableTask")) {
            //如果一个分表也很大，再分页
            String tableName = (String)task;
            Pair<Long, Long> idPair = queryMinAndMaxId(tableName);
            long minId = idPair.getFirst();
            long maxId = idPair.getSecond();
            List<PageTask> tasks = Lists.newArrayList();
            int step = (int) ((maxId - minId) / PAGE_SIZE); //计算分页数量
            for (long i = minId; i < maxId; i+=step) {
                tasks.add(new PageTask(tableName, i, (i+step > maxId ? maxId : i+step)));
            }
            return map(tasks, "PageTask");
        } else if (taskName.equals("PageTask")) {
            PageTask pageTask = (PageTask)task;
            String tableName = pageTask.getTableName();
            long startId = pageTask.getStartId();
            long endId = pageTask.getEndId();
            List<Record> records = queryRecord(tableName, startId, endId);
            //TODO handle records
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
```

```
    }
    private List<String> getDbList() {
        List<String> dbList = Lists.newArrayList();
        //TODO 返回分库列表
        return dbList;
    }
    private List<String> getTableList(String dbName) {
        List<String> tableList = Lists.newArrayList();
        //TODO 返回分表列表
        return tableList;
    }
    private Pair<Long, Long> queryMinAndMaxId(String tableName) {
        //TODO select min(id),max(id) from [tableName]
        return new Pair<Long, Long>(1L, 10000L);
    }
    private List<Record> queryRecord(String tableName, long startId, long endId) {
        List<Record> records = Lists.newArrayList();
        //TODO select * from [tableName] where id>=[startId] and id<[endId]
        return records;
    }
}
```

6.7.4. MapReduce模型

MapReduce模型是Map模型的扩展，新增Reduce接口，需要实现MapReduceJobProcessor。

背景信息

- MapReduce模型只有一个Reduce，所有子任务完成后会执行Reduce方法，可以在Reduce方法中返回该任务示例的执行结果，作为工作流的上下游数据传递。如果有子任务失败，Reduce不会执行。Reduce失败，整个任务示例也失败。
- MapReduce模型还能处理所有子任务的结果。子任务通过 `return ProcessResult(true, result)` 返回结果（例如返回订单号），Reduce的时候，可以通过context获取所有子任务的结果，进行相应的处理。

MapReduce模型的原理和最佳实践，请参见SchedulerX 2.0 分布式计算原理和最佳实践。

SchedulerX 2.0支持MapReduce模型的详细信息，请参见SchedulerX 2.0 支持 MapReduce 模型。

注意事项

- 所有子任务结果会缓存在Master节点，内存压力较大，建议子任务个数和Result不要太大。
- SchedulerX不保证子任务绝对执行一次，在特殊条件下会Failover，可能会导致子任务重复执行，需要业务方自己实现幂等。

接口

接口	解释	是否必选
`public ProcessResult process(JobContext context) throws Exception;`	每个子任务执行业务的入口，需要从context里获取taskName，自己判断是哪个子任务，进行相应的逻辑处理。执行完成后，需要返回ProcessResult。	是
`public ProcessResult map(List<? extends Object> taskList, String taskName);`	执行map方法可以把一批子任务分布式到多台机器上执行，可以map多次。如果taskList是空，返回失败。执行完成后，需要返回ProcessResult。	是

接口	解释	是否必选
`public ProcessResult reduce(JobContext context);`	无	是
`public void kill(JobContext context);`	前端kill任务会触发该方法，需要用户自己实现如何中断业务。	否

执行方式

- 并行计算：最多支持300任务，有子任务列表。

> 注意　秒级别任务不要选择并行计算。

- 内存网格：基于内存计算，最多支持50,000以下子任务，速度快。
- 网格计算：基于文件计算，最多支持1,000,000子任务。

高级配置

任务管理高级配置参数说明如下：

参数	适用的执行模式	解释	默认值
实例失败重试次数	通用	任务运行失败自动重试的次数。	0
实例失败重试间隔	通用	每次失败重试的间隔。单位：秒。	30
实例并发数	通用	同一个Job同一时间运行的实例个数。1表示不允许重复执行。	1
子任务单机并发数	• 并行计算 • 内存网格 • 网格计算	分布式模型，单台机器并发消费子任务的个数。	5
子任务失败重试次数	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的次数。	0
子任务失败重试间隔	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的间隔。单位：秒。	0
子任务分发方式	• 并行计算 • 内存网格 • 网格计算	• **推模型**：每台机器平均分配子任务。 • **拉模型**： 每台机器主动拉取子任务，没有木桶效应。拉取过程中，所有子任务会缓存在Master节点，对内存有压力，建议子任务数不超过10,000。	推模型

参数	适用的执行模式	解释	默认值
子任务单次拉取数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点每次向Master节点拉取多少个子任务。	5
子任务队列容量（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点缓存子任务的队列大小。	10
子任务全局并发数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	分布式拉模型支持全局子任务并发数，可以进行限流。	1,000

发送500条消息的Demo示例（适用于MapReduce模型）

```
@Component
public class TestMapReduceJobProcessor extends MapReduceJobProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        int dispatchNum=500;
        if (isRootTask(context)) {
            System.out.println("start root task");
            List<String> msgList = Lists.newArrayList();
            for (int i = 0; i <= dispatchNum; i++) {
                msgList.add("msg_" + i);
            }
            return map(msgList, "Level1Dispatch");
        } else if (taskName.equals("Level1Dispatch")) {
            String task = (String)context.getTask();
            System.out.println(task);
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
    }
    @Override
    public ProcessResult reduce(JobContext context) throws Exception {
        return new ProcessResult(true, "TestMapReduceJobProcessor.reduce");
    }
}
```

处理单表数据的Demo示例（适用于Map或MapReduce模型）

```
@Component
public class ScanSingleTableJobProcessor extends MapJobProcessor {
    @Service
    private XXXService xxxService;
    private final int PAGE_SIZE = 500;
    static class PageTask {
        private long startId;
        private long endId;
        public PageTask(long startId, long endId) {
            this.startId = startId;
            this.endId = endId;
        }
        public long getStartId() {
            return startId;
        }
        public long getEndId() {
            return endId;
        }
    }
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String tableName = context.getJobParameters(); //多个Job后端代码可以一致，通过控制台配置Job参数表示表名。
        String taskName = context.getTaskName();
        Object task = context.getTask();
        if (isRootTask(context)) {
            Pair<Long, Long> idPair = queryMinAndMaxId(tableName);
            long minId = idPair.getFirst();
            long maxId = idPair.getSecond();
            List<PageTask> tasks = Lists.newArrayList();
            int step = (int) ((maxId - minId) / PAGE_SIZE); //计算分页数量
            for (long i = minId; i < maxId; i+=step) {
                tasks.add(new PageTask(i, (i+step > maxId ? maxId : i+step)));
            }
            return map(tasks, "PageTask");
        } else if (taskName.equals("PageTask")) {
            PageTask pageTask = (PageTask)task;
            long startId = pageTask.getStartId();
            long endId = pageTask.getEndId();
            List<Record> records = queryRecord(tableName, startId, endId);
            //TODO handle records
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
    }
    private Pair<Long, Long> queryMinAndMaxId(String tableName) {
        //TODO select min(id),max(id) from [tableName]
        return new Pair<Long, Long>(1L, 10000L);
    }
    private List<Record> queryRecord(String tableName, long startId, long endId) {
        List<Record> records = Lists.newArrayList();
        //TODO select * from [tableName] where id>=[startId] and id<[endId]
        return records;
    }
}
```

处理分库分表数据的Demo示例（适用于Map或MapReduce模型）

```
@Component
public class ScanShardingTableJobProcessor extends MapJobProcessor {
    @Service
    private XXXService xxxService;
    private final int PAGE_SIZE = 500;
    static class PageTask {
        private String tableName;
        private long startId;
        private long endId;
        public PageTask(String tableName, long startId, long endId) {
            this.tableName = tableName;
            this.startId = startId;
            this.endId = endId;
        }
        public String getTableName() {
            return tableName;
        }
        public long getStartId() {
            return startId;
        }
        public long getEndId() {
            return endId;
        }
    }
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        Object task = context.getTask();
        if (isRootTask(context)) {
            //先分库
            List<String> dbList = getDbList();
            return map(dbList, "DbTask");
        } else if (taskName.equals("DbTask")) {
            //根据分库去分表
            String dbName = (String)task;
            List<String> tableList = getTableList(dbName);
            return map(tableList, "TableTask");
        } else if (taskName.equals("TableTask")) {
            //如果一个分表也很大，再分页
            String tableName = (String)task;
            Pair<Long, Long> idPair = queryMinAndMaxId(tableName);
            long minId = idPair.getFirst();
            long maxId = idPair.getSecond();
            List<PageTask> tasks = Lists.newArrayList();
            int step = (int) ((maxId - minId) / PAGE_SIZE); //计算分页数量
            for (long i = minId; i < maxId; i+=step) {
                tasks.add(new PageTask(tableName, i, (i+step > maxId ? maxId : i+step)));
            }
            return map(tasks, "PageTask");
        } else if (taskName.equals("PageTask")) {
            PageTask pageTask = (PageTask)task;
            String tableName = pageTask.getTableName();
            long startId = pageTask.getStartId();
            long endId = pageTask.getEndId();
            List<Record> records = queryRecord(tableName, startId, endId);
            //TODO handle records
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
```

```
    }
    private List<String> getDbList() {
        List<String> dbList = Lists.newArrayList();
        //TODO 返回分库列表
        return dbList;
    }
    private List<String> getTableList(String dbName) {
        List<String> tableList = Lists.newArrayList();
        //TODO 返回分表列表
        return tableList;
    }
    private Pair<Long, Long> queryMinAndMaxId(String tableName) {
        //TODO select min(id),max(id) from [tableName]
        return new Pair<Long, Long>(1L, 10000L);
    }
    private List<Record> queryRecord(String tableName, long startId, long endId) {
        List<Record> records = Lists.newArrayList();
        //TODO select * from [tableName] where id>=[startId] and id<[endId]
        return records;
    }
}
```

处理50条消息并且返回子任务结果由Reduce汇总的Demo示例（适用于MapReduce模型）

```
@Component
public class TestMapReduceJobProcessor extends MapReduceJobProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        int dispatchNum = 50;
        if (context.getJobParameters() != null) {
            dispatchNum = Integer.valueOf(context.getJobParameters());
        }
        if (isRootTask(context)) {
            System.out.println("start root task");
            List<String> msgList = Lists.newArrayList();
            for (int i = 0; i <= dispatchNum; i++) {
                msgList.add("msg_" + i);
            }
            return map(msgList, "Level1Dispatch");
        } else if (taskName.equals("Level1Dispatch")) {
            String task = (String)context.getTask();
            Thread.sleep(2000);
            return new ProcessResult(true, task);
        }
        return new ProcessResult(false);
    }
    @Override
    public ProcessResult reduce(JobContext context) throws Exception {
        for (Entry<Long, String> result : context.getTaskResults().entrySet()) {
            System.out.println("taskId:" + result.getKey() + ", result:" + result.getValue());
        }
        return new ProcessResult(true, "TestMapReduceJobProcessor.reduce");
    }
}
```

6.7.5. 多语言版本分片模型

SchedulerX可以对多重任务进行调度（定时、编排、重刷历史数据等），提供Java、Python、Shell和Go等多语言分片模型，帮助您处理大数据业务需求。

背景信息

分片模型主要包含静态分片和动态分片。

- 静态分片：主要场景是处理固定的分片数，例如分库分表中固定1024张表，需要若干台机器分布式去处理。
- 动态分片：主要场景是分布式处理未知数据量的数据，例如一张大表在不停变更，需要分布式跑批。主流的框架为SchedulerX提供的MapReduce模型，暂时还没有对外开源。

功能特性

多语言版本分片模型还具有以下特性。

- 兼容elastic-job的静态分片模型。
- 支持Java、Python、Shell、Go四种语言。
- 高可用：分片模型基于Map模型开发，可以继承Map模型高可用特性，即某台worker执行过程中发生异常，master worker会把分片failover到其它slave节点执行。
- 流量控制：分片模型基于Map模型开发，可以继承Map模型流量控制特性，即可以控制单机子任务并发度。例如有1000个分片，一共10台机器，可以控制最多5个分片并发跑，其它在队列中等待。
- 分片自动失败重试：分片模型基于Map模型开发，可以继承Map模型子任务失败自动重试特性。

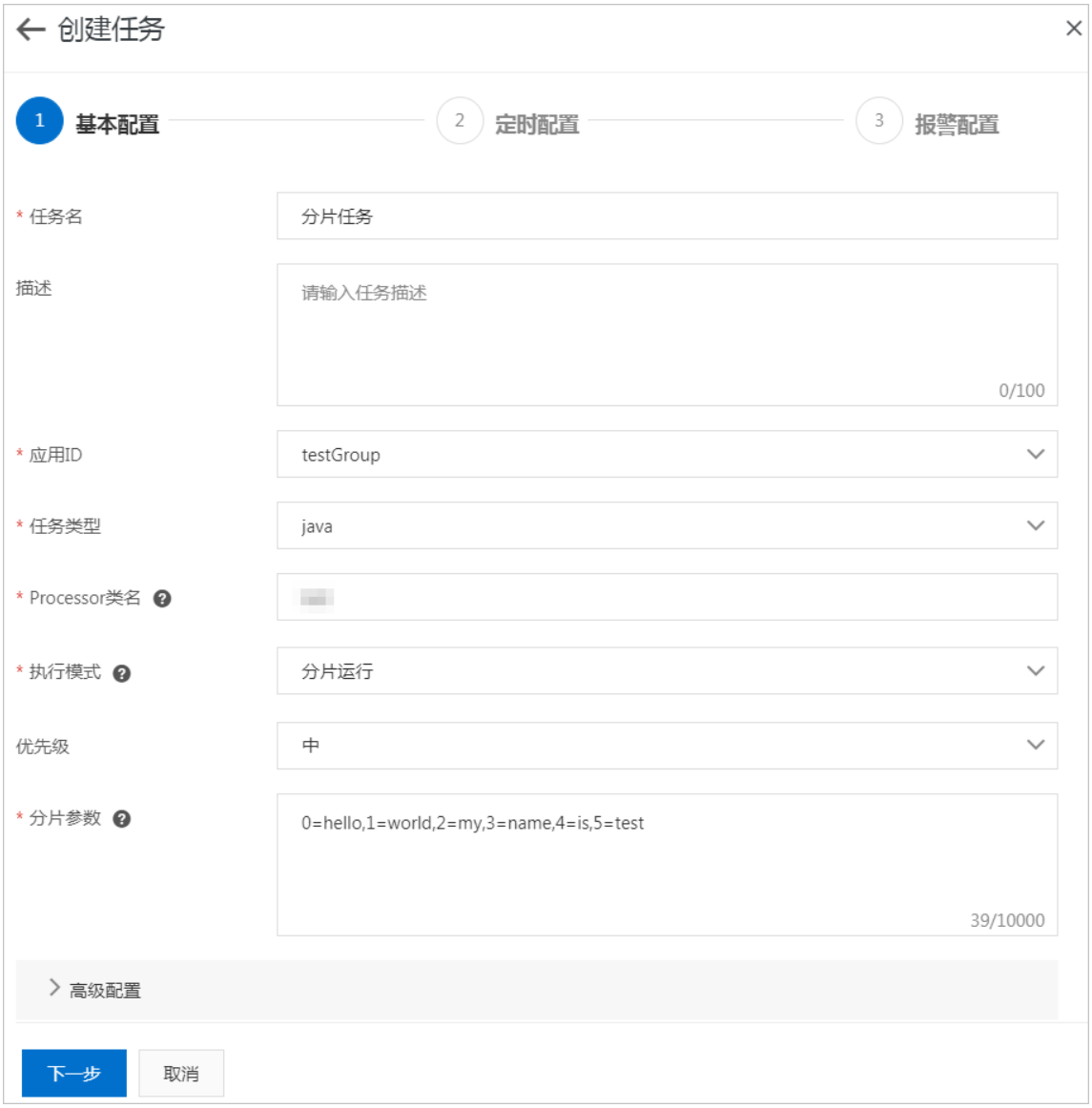
可用性和流量控制可以在创建任务时的高级配置中设置，详情请参见创建调度任务和任务管理高级配置参数说明。

> **说明** 只有1.1.0及以上版本客户端才支持多语言版本的分片模型。

Java分片任务

1. 登录EDAS控制台。
2. 在顶部菜单栏选择地域。
3. 在左侧导航栏选择**组件中心 > 分布式任务调度（新）**，然后单击**任务管理**。
4. 在**任务管理**页面，选择目标命名空间，在页面左上角单击**创建任务**。
5. 在**创建任务**面板的**基本配置**配置向导页面的**执行模式**列表选择**分片运行**，并设置**分片参数**。

   分片参数之间以半角逗号（,）或换行分隔，例如 `分片号1=分片参数1,分片号2=分片参数2,...` 。



6. 在应用程序代码中继承 `JavaProcessor`，通过 `JobContext.getShardingId()` 获取分片号，通过 `JobContext.getShardingParameter()` 获取分片参数。

   示例：

```
@Component
public class HelloWorldProcessor extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("分片id=" + context.getShardingId() + ", 分片参数=" + context.getShardingParameter());
        return new ProcessResult(true);
    }
}
```

7. 在执行列表页面查看分片详情。

Python分片任务

Python应用想使用分布式跑批，只需要安装Agent。脚本可以由SchedulerX维护。

1. 下载SchedulerX的Agent，并通过Agent部署脚本任务。
2. 在SchedulerX中创建Python分片任务，更多信息，请参见创建调度任务。

   `sys.argv[1]` 为分片号，`sys.argv[2]` 为分片参数。
   分片参数之间以半角逗号（,）或换行分隔，例如 `分片号1=分片参数1,分片号2=分片参数2,...` 。

3. 在执行列表页面查看分片详情。

Shell和Go分片任务

Shell和Go版本的分片任务和Python类似，创建步骤，请参见Python分片任务。

6.8. 高级特性

6.8.1. 如何接入日志服务

阿里巴巴分布式任务调度系统SchedulerX 2.0的日志服务，您不需要修改一行代码，只需要增加一个Log4j或Logback的配置，即可在控制台看到每次任务调度（包括分布式任务）的业务日志，方便排查问题。

前提条件

- （可选）创建命名空间。
- 创建应用。
- 应用升级为专业版并开通日志服务。具体操作，请参见如何升级为专业版。

> **说明** 开通日志服务后，当前日志最多可保留2周，超过2周的日志会被清理。

接入配置

1. 将SchedulerX客户端升级到1.4.0以上版本。

   以Spring Boot Starter为例，在应用程序的 `pom.xml` 文件中添加SchedulerxWorker依赖。

```
<dependency>
    <groupId>com.aliyun.schedulerx</groupId>
    <artifactId>schedulerx2-spring-boot-starter</artifactId>
    <version>1.4.0</version>
</dependency>
```

2. 配置Log Appender采集日志服务。
   - Log4j2 Appender
     a. `log4j2.xml` 增加一个名为 `SchedulerxLog4j2Appender` 的Appender。

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="off">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout
                pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %m%n" />
        </Console>
        <SchedulerxLog4j2Appender name="schedulerxLog"
            timeFormat="yyyy-MM-dd'T'HH:mmZ"
            timeZone="UTC"
            ignoreExceptions="true">
        <PatternLayout pattern="%d %-5level [%thread] %logger{0}: %msg"/>
    </SchedulerxLog4j2Appender>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console" />
        </Root>
        <Logger name="schedulerx" level="info" additivity="false">
            <AppenderRef ref="schedulerxLog" />
        </Logger>
    </Loggers>
</Configuration>
```

b. 业务代码使用原生log4j2打印日志。

```
package com.hxm.test.processor;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.stereotype.Component;
@Component
public class HelloWorldJob3 extends JavaProcessor {
    private static final Logger LOGGER = LogManager.getLogger("schedulerx");
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        LOGGER.info("hello HelloWorldJob3");
        return new ProcessResult(true);
    }
}
```

- Log4j Appender

  `log4j.properties` 增加一个Appender:

  ```
  log4j.appender.schedulerxLog=com.alibaba.schedulerx.worker.log.appender.SchedulerxLog4jAppender
  ```

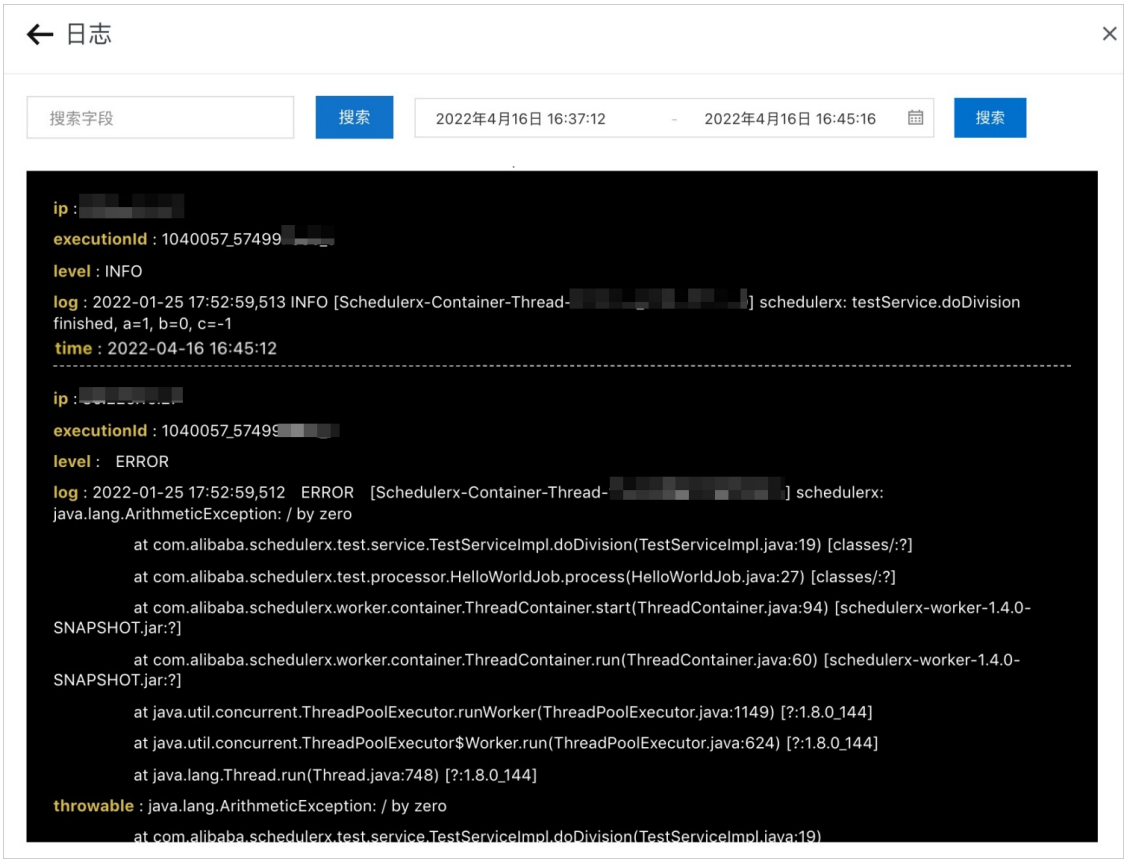
- Logback Appender

  `logback.xml` 增加一个Appender:

  ```
  <appender name="schedulerxLog" class="com.alibaba.schedulerx.worker.log.appender.SchedulerxLogbackAppender">
      <timeFormat>yyyy-MM-dd'T'HH:mmZ</timeFormat>
      <timeZone>UTC</timeZone>
  </appender>
  ```

3. 查看输出的日志。
   i. 登录分布式任务调度平台。
   ii. 在左侧导航栏单击**任务管理**。
   iii. 在任务列表中找到目标任务，在目标任务的操作列选择 > **历史记录**。

iv. 在**任务示例记录**页面单击目标任务操作列的**日志**。



日志中关键词的定义如下：

- `ip`：打印该日志的执行机器。
- `executionId`：本次任务实例的执行ID，格式为 `${jobId}_${jobInstanceId}_${taskId}` 。
- `level`：日志的级别。
- `log`： 日志的信息。
- `throwable`：应用抛出异常时，会打印该字段。

使用场景示例

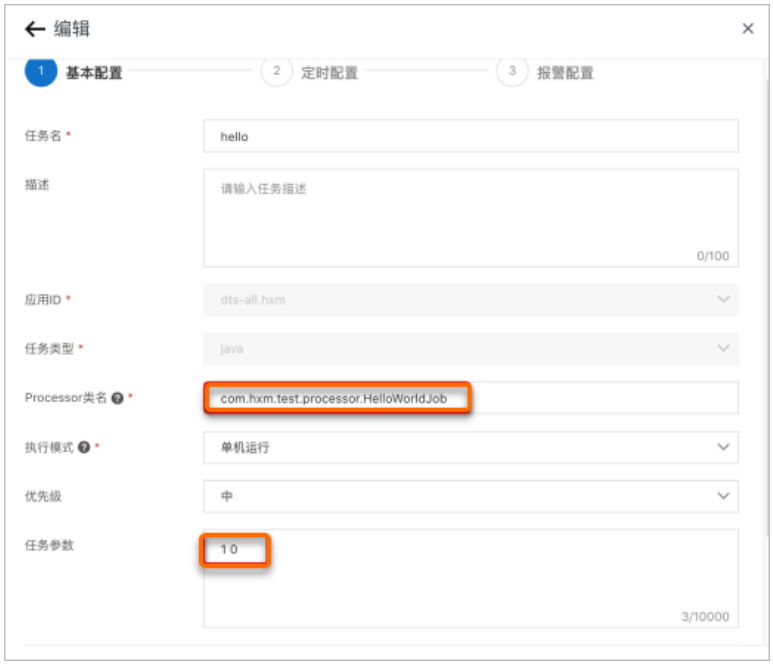
- 查询业务失败的原因
  SchedulerX 2.0的日志服务，可以收集任务的执行日志和异常，包括Service的日志都可以收集。
  i. 新建任务代码，配置使用SchedulerX打印日志。

```
package com.hxm.test.processor;
import com.alibaba.schedulerx.test.service.TestService;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.JavaProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class HelloWorldJob extends JavaProcessor {
    //使用SchedulerX收集日志
    private static final Logger LOGGER = LogManager.getLogger("schedulerx");
    @Autowired
    private TestService testService;
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String parameters = context.getJobParameters();
        String tokens[] = parameters.split(" ");
        int a = Integer.valueOf(tokens[0]);
        int b = Integer.valueOf(tokens[1]);
        int c = testService.doDivision(a, b);
        LOGGER.info("testService.doDivision finished, a={}, b={}, c={}", a, b, c);
        if (c < 0) {
            return new ProcessResult(false, "result=" + c);
        }
        return new ProcessResult(true);
    }
}
```

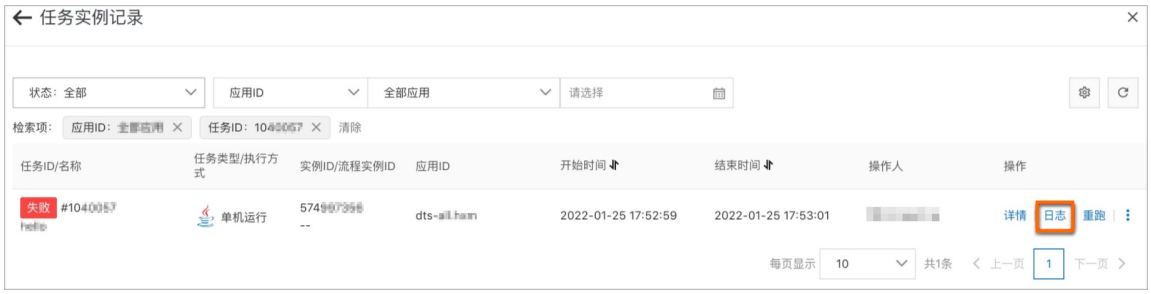
ii. 新建Service业务代码，配置使用SchedulerX打印日志。

```
package com.hxm.test.service;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.stereotype.Service;
@Service("testService")
public class TestServiceImpl implements TestService {
    //使用SchedulerX收集日志
    private static final Logger LOGGER = LogManager.getLogger("schedulerx");
    @Override
    public int doDivision(int a, int b) {
        try {
             LOGGER.info("start to do division c = " + a + "/" + b);
             int c = a/b;
             LOGGER.info("c=" + c);
             return c;
        } catch (Exception e) {
            LOGGER.error("", e);
        }
        return -1;
    }
}
```

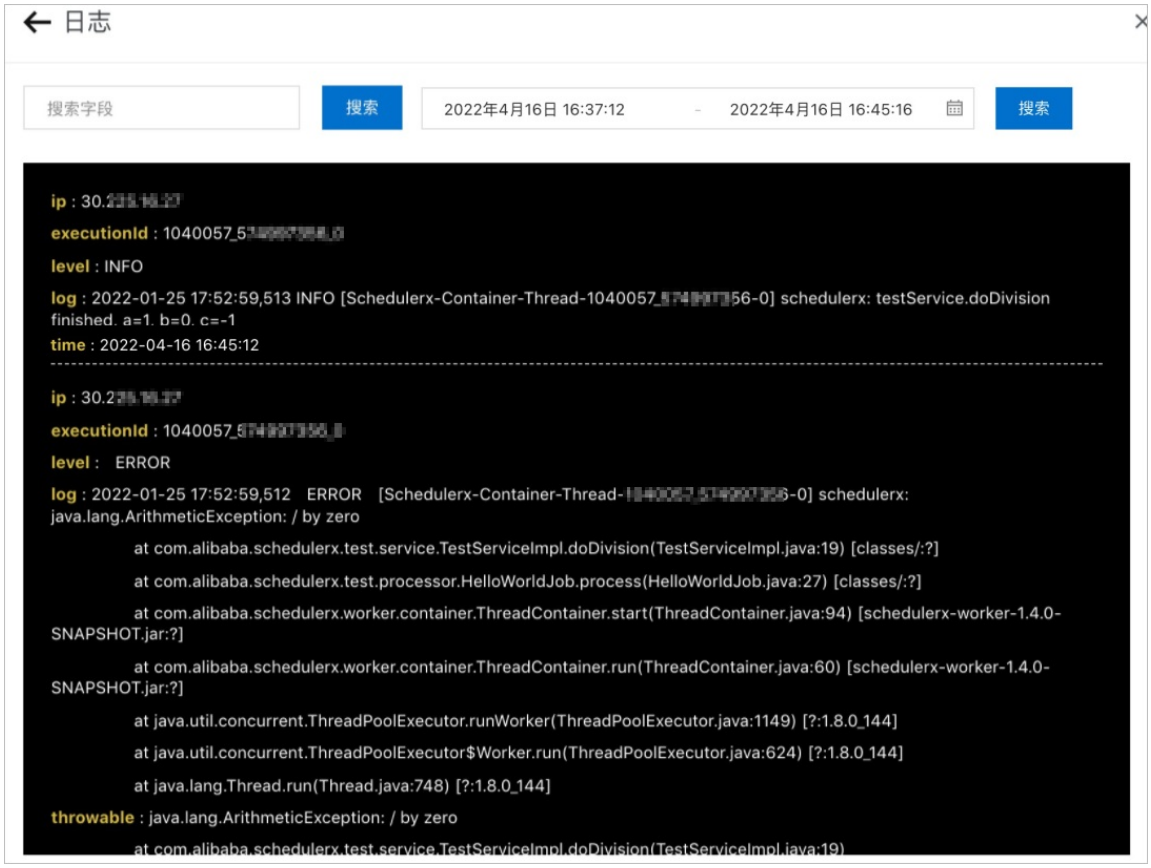
iii. 控制台配置任务。
根据代码逻辑可知1除以0肯定会抛出异常。



iv. 任务运行一次后，通过任务实例列表，查看日志。



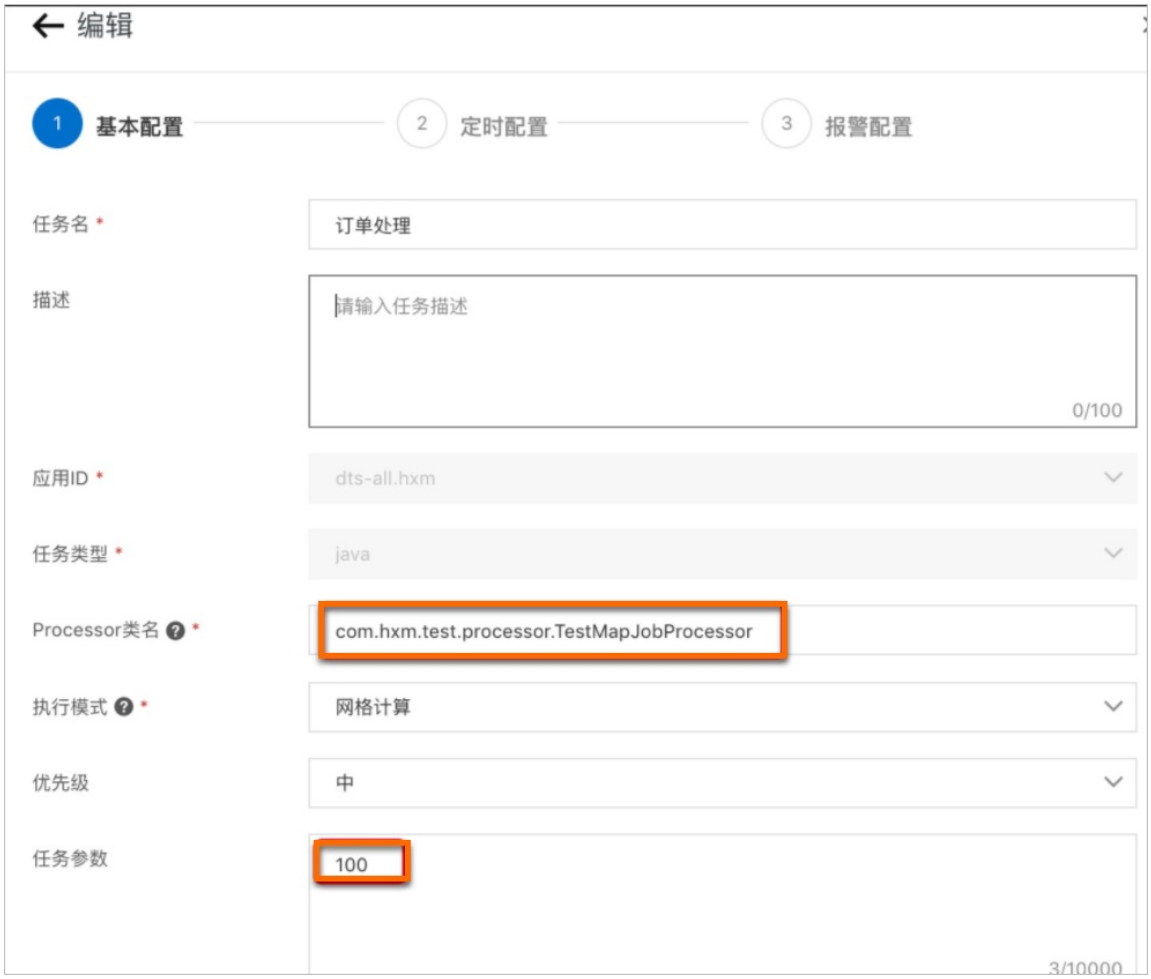
v. 根据日志内容，可得出失败的原因是TestServiceImpl抛出了除0的异常。



- 查询分布式任务失败原因
  SchedulerX 2.0的分布式任务用来执行批量任务，例如某个任务批次执行失败了，您想查询具体是哪个子任务失败时，可执行如下操作：
  i. 新建任务代码，配置使用SchedulerX打印日志。

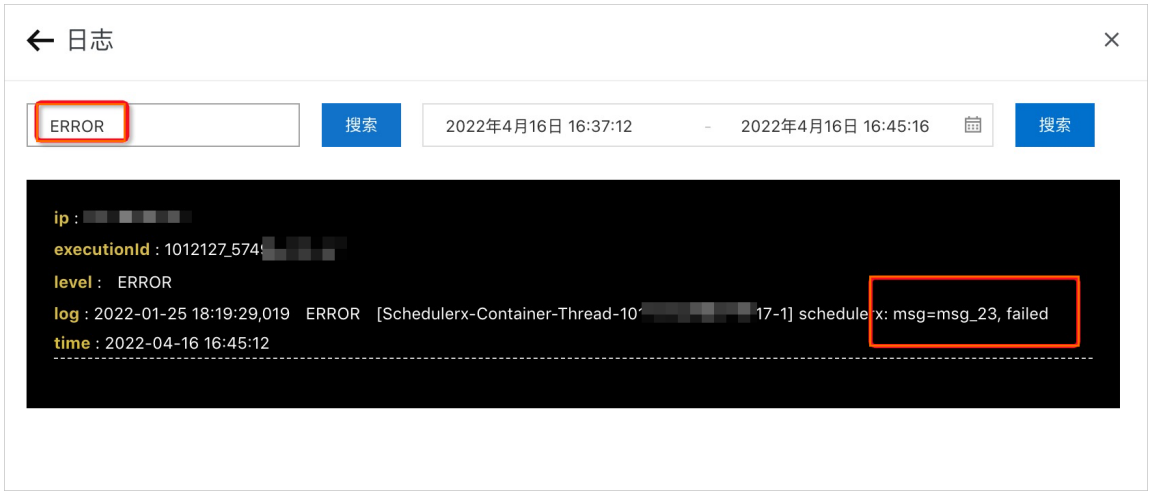
```
package com.hxm.test.processor;
import java.util.ArrayList;
import java.util.List;
import com.alibaba.schedulerx.worker.domain.JobContext;
import com.alibaba.schedulerx.worker.processor.MapJobProcessor;
import com.alibaba.schedulerx.worker.processor.ProcessResult;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
public class TestMapJobProcessor extends MapJobProcessor {
    //使用SchedulerX收集日志
    private static final Logger LOGGER = LogManager.getLogger("schedulerx");
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        String taskName = context.getTaskName();
        String parameter = context.getJobParameters();
        int dispatchNum = Integer.valueOf(parameter);
        if (isRootTask(context)) {
            LOGGER.info("start root task");
            List<String> msgList = new ArrayList<>();
            for (int i = 0; i <= dispatchNum; i++) {
                msgList.add("msg_" + i);
            }
            return map(msgList, "Level1Dispatch");
        } else if (taskName.equals("Level1Dispatch")) {
            String task = (String)context.getTask();
            if (task.equals("msg_23")) {
                LOGGER.error("msg={}, failed", task);
                return new ProcessResult(false);
            } else {
                LOGGER.info("msg={}, success", task);
            }
            return new ProcessResult(true);
        }
        return new ProcessResult(false);
    }
}
```

ii. 控制台配置任务并执行一次。

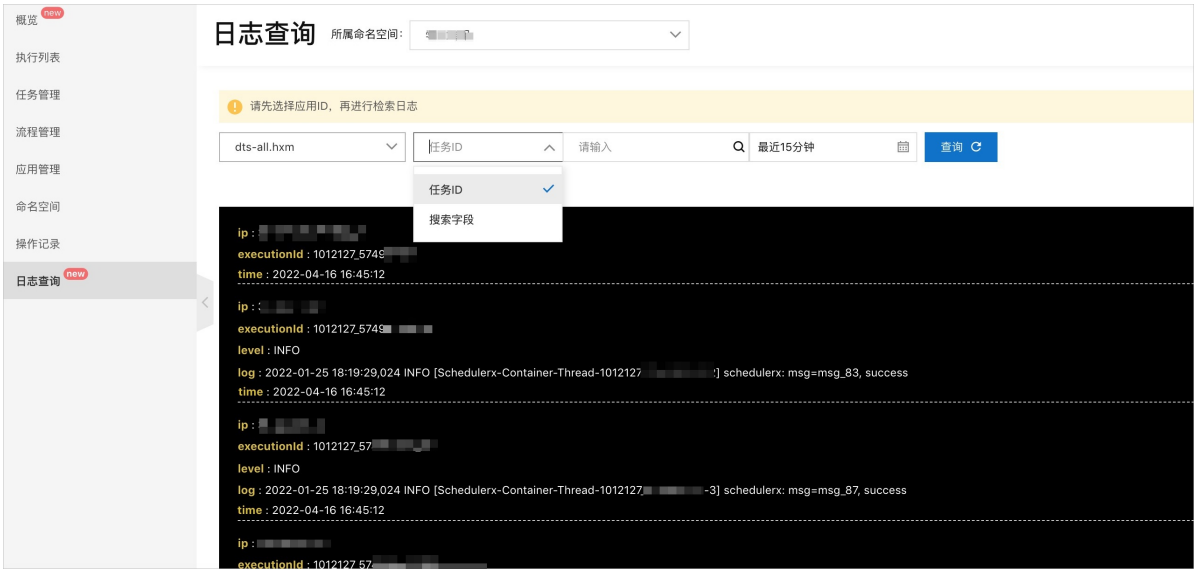


iii. 查看日志。具体操作，请参见查看日志。

iv. 在日志中搜索关键字，快速定位失败的任务及原因。



- 根据关键字查询历史日志
  任务实例的历史记录只保留最近60条。如果想排查历史上任务失败的原因，可以通过控制台左侧导航栏的**日志查询**来检索。**日志查询**支持根据任务ID、关键字和时间区间搜索。



6.8.2. 如何使单应用支持十万以上的定时任务

在SchedulerX中创建的应用默认最多只支持1,000个任务，而在某些场景下1,000个任务远远满足不了业务需求。另外，实际业务场景中，不同定时任务的调度时间可能不一样。本文介绍如何通过一系列配置，使单应用能够支持上万，甚至十万以上的定时任务，并且不同任务能够有独立的调度时间。

应用场景

常见应用场景是每个定时任务调度时间都不一样，无法通过MapReduce分布式任务解决（MapReduce分布式任务每个子任务调度时间是一致的）。

- 物联网智能开关
  智能开关可以设置定时开启、关闭，是由用户自定义设置的，所以每个开关的定时时间是不一致的。可以给每一个开关设置一个定时单机任务。在物联网场景下，定时单机任务可能会达到万、甚至十万级别。
- 业务监控
  业务监控需要配置监控报警规则，一般每分钟轮询一次，符合规则则报警。大部分场景下使用一个MapReduce分布式任务是可以解决的，但是如果由于报警规则的复杂程度不同而导致执行时间差异较大时，有可能出现一个报警子任务没有执行完而阻塞整个任务下次调度。所以，可以给每个报警规则配置一个定时单机任务。当业务规模很大时，定时单机任务也可能达到万、甚至十万级别。
- 将SchedulerX作为底座
  当将SchedulerX作为底座，封装一层任务调度平台（通过POP API新建任务）提供给自己的公司使用时，任务量也有可能达到万、甚至十万级别。

操作步骤

使单应用能够支持十万以上的定时任务，需要完成以下两个步骤：

1. 联系SchedulerX技术支持人员，开启应用自动扩容。

   开启应用自动扩容后，单应用的任务数达到上限（1000）后，会自动感知并分裂出一个新的子应用。

2. 使用1.2.1及以上版本客户端接入SchedulerX，并开启共享ContainerPool功能。

   本文以Spring Boot应用为例进行介绍，客户端接入详情请参见Spring Boot应用接入SchedulerX。如果您使用其它类型的应用接入，请参见**快速入门 > 客户端快速接入**中的相关文档。

   i. 在应用的*pom.xml*文件中添加1.2.1及以上版本的客户端依赖。

      1.2.1以下版本客户端不支持共享ContainerPool，所以请在应用中添加1.2.1及以上版本客户端，或将客户端的依赖升级到1.2.1及以上版本。更多客户端版本信息请参见发布记录。

ii. 在应用的配置文件中添加共享ContainerPool配置。

```
spring.schedulerx2.shareContainerPool=true #开启所有任务共享线程池
spring.schedulerx2.sharePoolSize=128    #自定义共享线程池大小
```

> **说明** 如果未开启共享ContainerPool，每个任务触发都会新建一个线程池，客户端负载会因超负荷而发生异常。

6.8.3. 如何管理应用级别的资源和任务优先级

对于业务规模较大的应用而言，调度的稳定性和核心任务的时效性是至关重要的，需要应用级别的资管管理和任务优先级来保证。本文介绍如何管理应用级别的资源和任务优先级。

背景信息

一些第三方的资源管理系统（例如Mesos和Yarn），能够实现CPU和内存级别的资源管控，而您使用自己的Worker通过客户端接入SchedulerX，所以SchedulerX作为通用的任务调度平台，无法实现CPU和内存级别的管控，也无法通过第三方的资源管理系统进行管控，实现的是任务实例数量和优先级的管控。

应用场景

应用级别的资管和任务优先级管理主要适用于业务、数据规模较大的调度场景。

例如，一个数据平台的应用，每天夜里会执行成千上万的报表，如果没有资源管理，应用可能会因为超负荷而发生故障。同时，一些核心报表也可能会有极强的时效性，必须在某个时间前生成，会对任务的优先级有强烈的需求。SchedulerX提供了资源管理和任务优先级的功能。

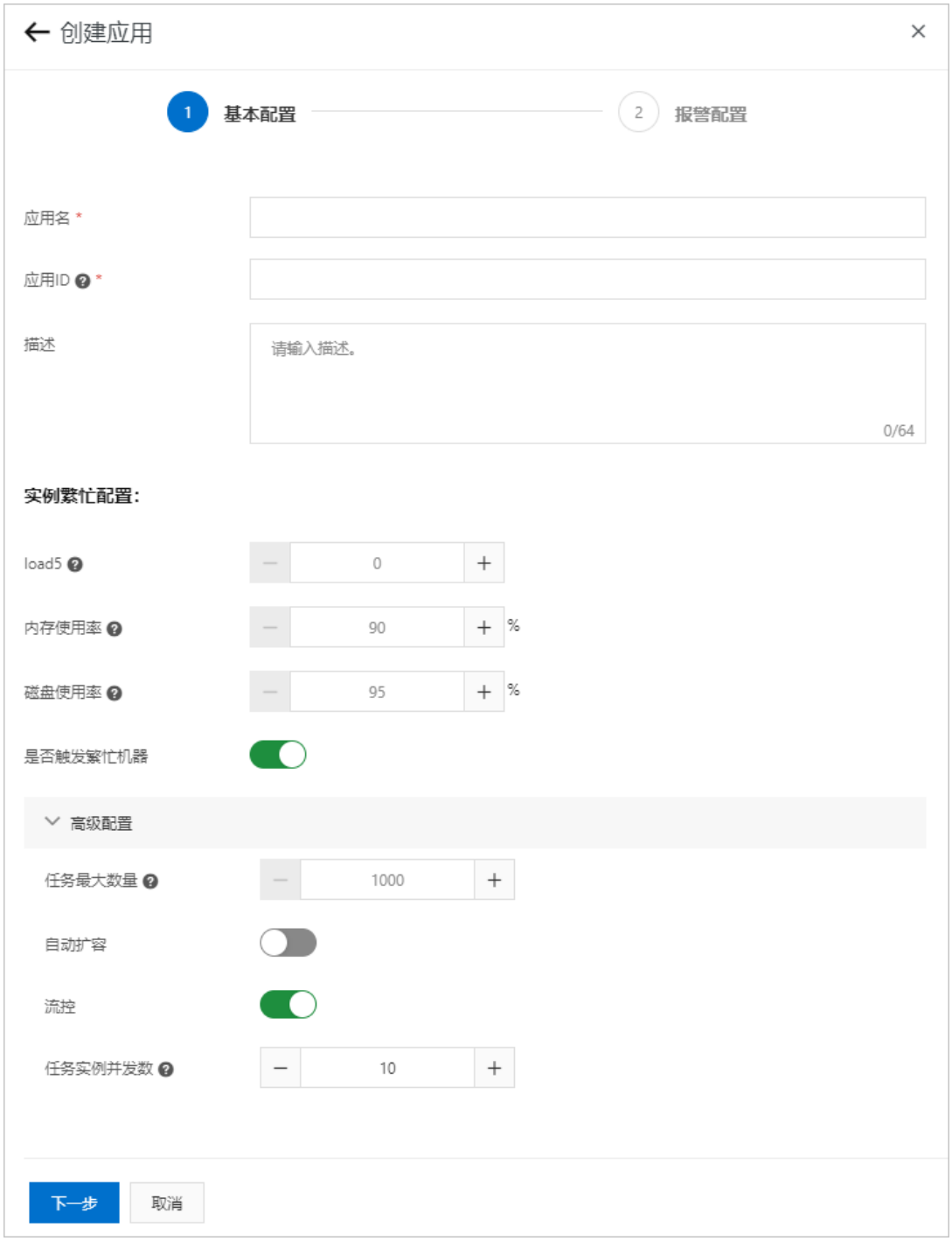
- 资源管理
  资源管理即管理应用的任务实例数量，例如在创建应用时，为该应用打开了流控开关，并将任务实例并发数设置为1。再在该应用下创建3个任务，A、B和C，每个任务运行一次，则任务A运行中，而任务B和C在池子中等待运行，而不会被丢弃。
- 任务优先级管理
  任务优先级是应用级别的，即优先级仅在单应用内生效，不影响其它应用的任务。同一个应用下，同时运行的优先级高的任务会被优先执行。

> **说明** 一个应用包含多个实例，不同优先级的任务被调度到不同实例执行，可能导致低优先级任务被优先执行。SchedulerX通过可抢占的优先级队列规避了这种可能性，并保证同时在池子中等待的高优先级任务被优先执行。详情请参见可抢占的优先级队列。

管理应用的任务实例资源

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4.
5.
6. 在**创建应用**面板中，单击**高级配置**，然后打开**流控**开关，并设置**任务实例并发数**。

   任务实例并发数即应用级别的任务队列，超过并发数的任务实例不会被丢弃，而是放到池子中等待执行。
   创建应用的其它参数，请参见应用管理。



管理应用的任务优先级

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏选择**任务管理**。
5. 在**任务管理**页面，选择目标命名空间，单击**创建任务**。
6. 在**创建任务**面板的**基本配置**配置向导页，设置**优先级**。

   创建任务的其它参数及后续步骤，请参见创建调度任务。

可抢占的优先级队列

一个应用包含多个实例，不同优先级的任务被调度到不同实例执行，可能导致低优先级任务被优先执行。SchedulerX通过可抢占的优先级队列规避了这种可能性，并保证同时在池子中等待的高优先级任务被优先执行。

1. 创建一个示例应用，并为该应用打开**流控**开关，并设置**任务实例并发数**为1，请参见管理应用的任务优先级。
2. 为该应用创建3个任务，优先级分别设置为高、中、低。请参见管理应用的任务优先级。
3. 在该应用的任务管理页面依次在中优先级任务、低优先级任务和高优先级任务的**操作**列单击**运行一次**。

4. 观察执行结果。
   i. 由于中优先级任务运行的时候，队列中时空的，所以中优先级直接被执行。

   ii. 中优先级任务执行完成后，高优先级任务会抢占低优先级任务的位置被优先执行。

6.8.4. 如何创建秒级调度任务

秒级任务适合对实时性要求比较高的业务，例如不停做轮询的准实时业务，通过内存网格和秒级调度，可以让您不停地处理海量的数据。本文将以一个实例介绍如何创建秒级调度任务。
SchedulerX的秒级别任务属于定时调度类型，适用于简单Java任务、分布式Java任务和脚本任务，以及各种执行方式。

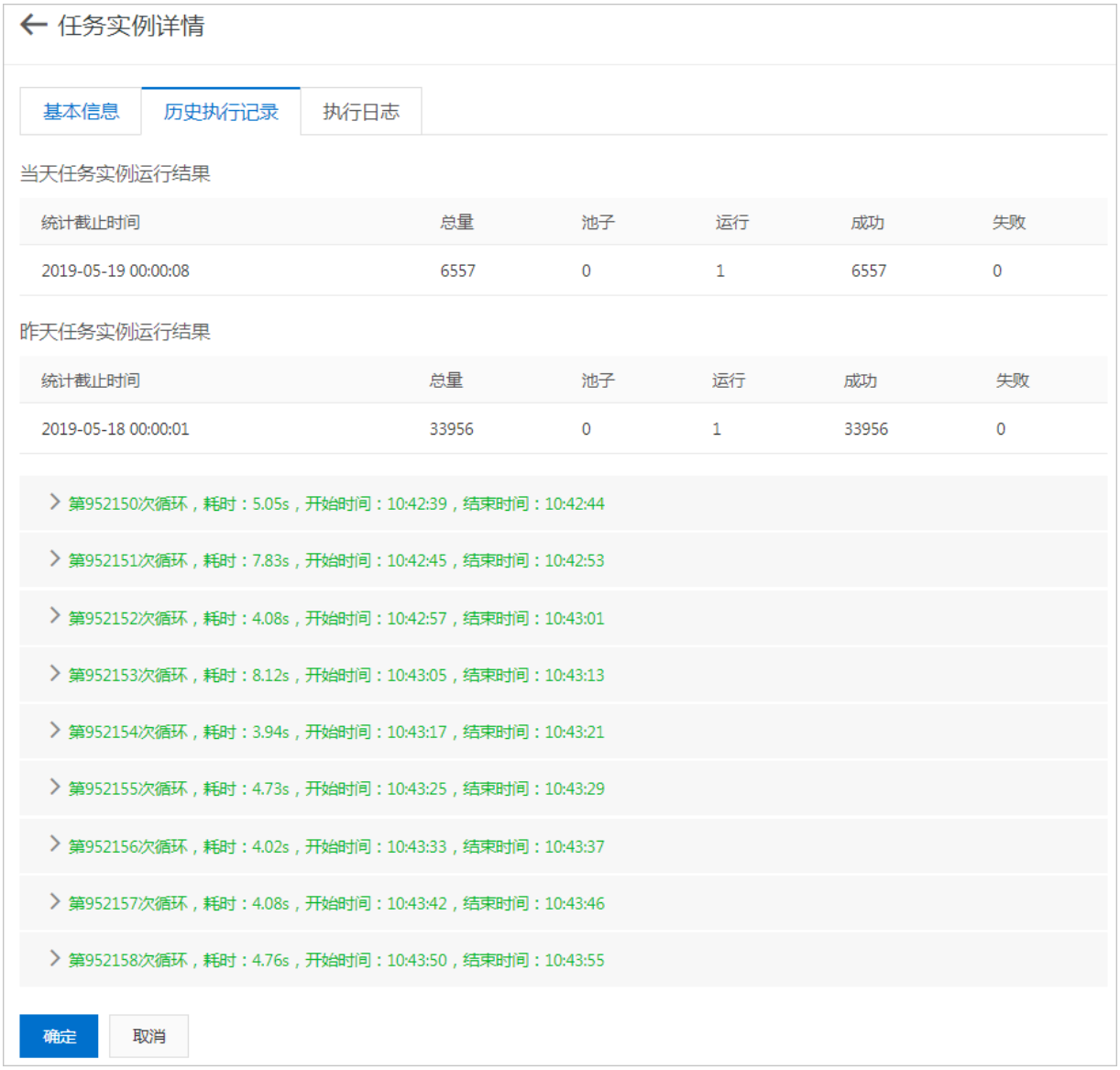
由于秒级调度属于定时调度，所以在定时配置步骤中请将**时间类型**设置为*second_delay*，并将**固定延迟**设置为*50*（秒）。创建调度任务的操作步骤请参见创建调度任务。

查看秒级调度任务详情。

查看任务实例详情的步骤请参见执行列表。

秒级任务在任务实例详情页面中会多包含**历史执行记录**页签，该页签记录了如下信息：

- **当天任务实例运行结果**：当天开始触发时间到现在为止，秒级任务调度总次数以及成功、失败次数。
- **昨天任务实例运行结果**：昨天触发时间到昨天结束，秒级任务昨天调度执行次数以及成功、失败次数。
- 最近10次运行结果。包含每次循环运行的子任务详情，包括每级分发的子任务总数及执行的成功失败数。



为秒级调度任务设置告警

- 秒级任务执行失败告警，这个告警指的是首次触发失败告警，首次触发成功后，秒级调度失败不会报触发失败告警。
- 秒级任务连续失败告警，如果您的任务首次触发成功后，后续秒级调度连续失败超过10次，则会收到告警。

上面两个告警都需要用户在任务告警配置中打开失败报警开关。

6.8.5. 如何通过工作流进行上下游数据传递

SchedulerX提供的工作流功能可以对多个任务进行编排，同时还支持上下游任务间的数据传递，让您的业务更加的简单易用。本文将以3个调度任务为例介绍如何通过工作流进行上下游任务间的数据传递。

背景信息

当前只有简单Java任务支持数据传递，分布式Java任务请使用MapReduce模型进行数据传递，详情请参见MapReduce模型。

操作步骤

1. 在三个应用中分别实现任务调度类JobProcessor A、JobProcessor B和JobProcessor C。
   - JobProcessor A

```
@Component
public class TestSimpleJobA extends JavaProcessor {
@Override
public ProcessResult process(JobContext context) throws Exception {
System.out.println("TestSimpleJobA " + DateTime.now().toString("yyyy-MM-dd HH:mm:ss"));
return new ProcessResult(true, String.valueOf(1));
}
}
```

- JobProcessor B

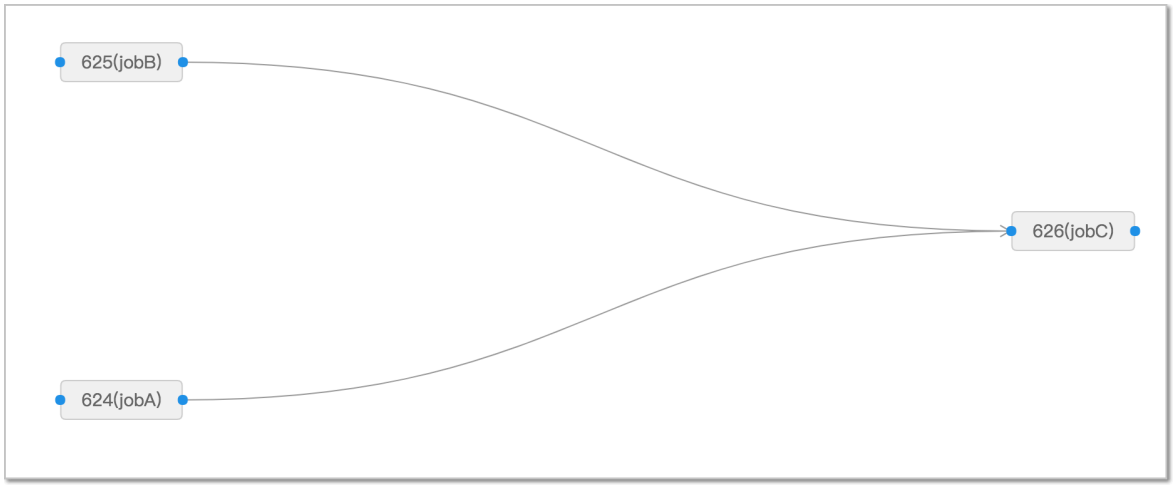
```
@Component
public class TestSimpleJobB extends JavaProcessor {
@Override
public ProcessResult process(JobContext context) throws Exception {
System.out.println("TestSimpleJobB " + DateTime.now().toString("yyyy-MM-dd HH:mm:ss"));
return new ProcessResult(true, String.valueOf(2));
}
}
```

- JobProcessor C

```
@Component
public class TestSimpleJobC extends JavaProcessor {
@Override
public ProcessResult process(JobContext context) throws Exception {
List<JobInstanceData> upstreamDatas = context.getUpstreamData();
int sum = 0;
for (JobInstanceData jobInstanceData : upstreamDatas) {
      System.out.println("jobName=" + jobInstanceData.getJobName()
          + ", data=" + jobInstanceData.getData());
      sum += Integer.valueOf(jobInstanceData.getData());
}
System.out.println("TestSimpleJobC sum=" + sum);
return new ProcessResult(true, String.valueOf(sum));
}
}
```

2. 将这三个应用部署到EDAS。
3. 分别为这三个应用创建任务分组和调度任务jobA、jobB和jobC，详情请参见应用管理和创建调度任务。
4. 创建工作流，并导入这3个调度任务，详情请参见创建工作流。

   创建好的工作流如下图所示：



5. 在**流程管理**页面具体流程的**操作**列单击**更多**，然后在下拉列表中选择**运行一次**。

执行结果

返回**工作流详情**页面，右键单击jobA、jobB和jobC，在快捷菜单中选择**详情**，查看任务实例详情。
可以看到jobA的**结果或错误信息**为1，和JobProcessor A一致。

同样，查看jobB的结果或错误信息为2，也和JobProcessor B一致。而jobC的结果为3（1+2），即上游任务jobA和jobB将数据传递给了jobC，和JobProcessor C的代码一致。

在控制台能看到同样的打印信息。

```
jobName=jobB, data=2
jobName=jobA, data=1
TestSimpleJobC sum=3
```

6.8.6. 如何设置数据时间

SchedulerX可以处理有数据状态的任务，您可以通过数据时间处理非任务执行时间的数据。

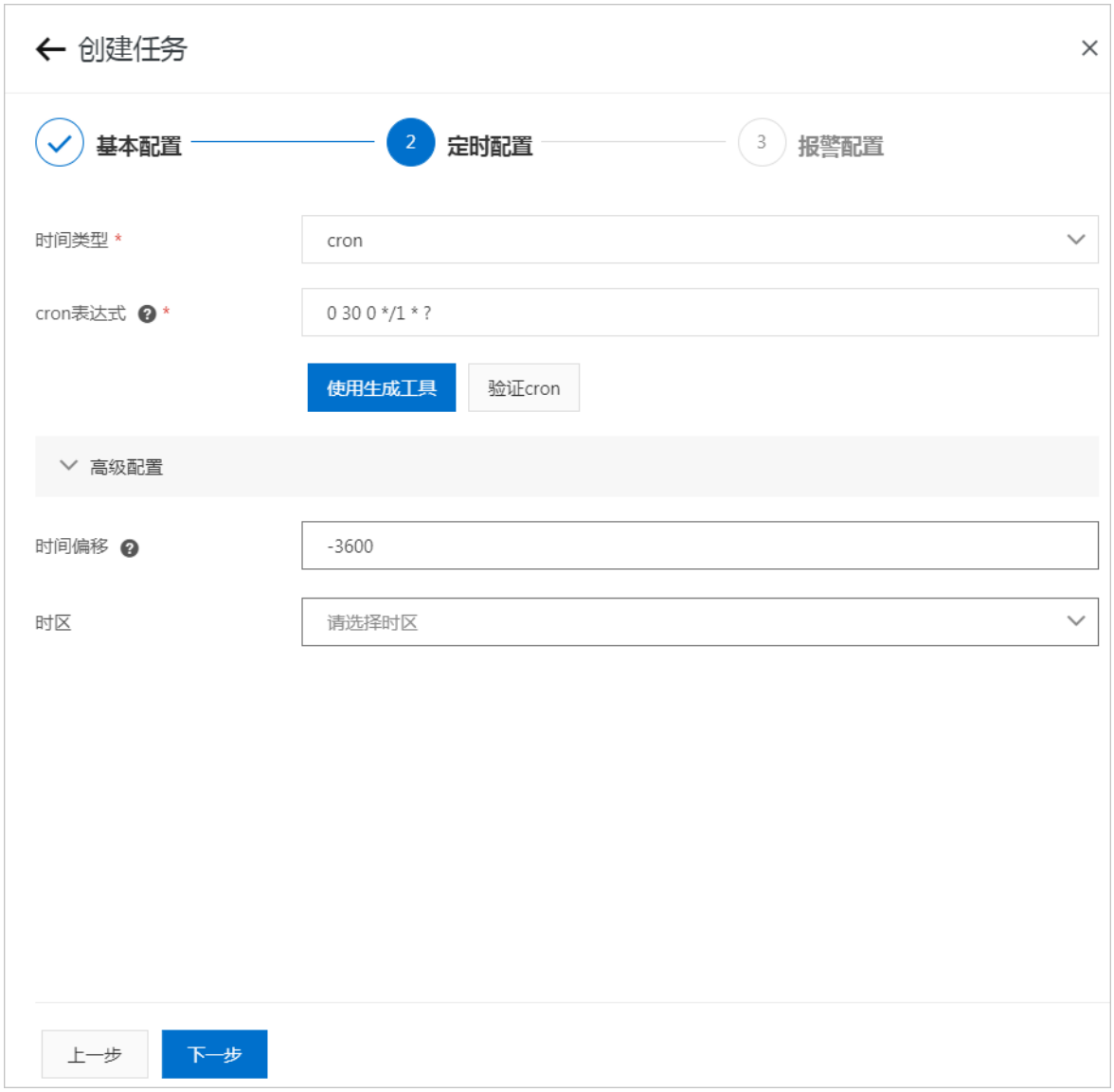
操作步骤

例如一个任务在每天00:30运行，但是实际上要处理前一天的数据，即数据时间需要在任务时间的基础上，向前偏移一小时。

1. 在客户端中接入SchedulerX，详情请参见快速入门章节，并实现数据时间。

```
public class TestHelloJob extends JavaProcessor {
    @Override
    public ProcessResult process(JobContext context) throws Exception {
        System.out.println("hello schedulerx2.0");
        System.out.println("dataTime=" + context.getDataTime().toString("yyyy-MM-dd HH:mm:ss"));
        return new ProcessResult(true);
    }
}
```

2. 在控制台创建任务，详情请参见创建调度任务。并在**定时配置**中设置**时间偏移** *-3600*（单位：秒），即向前偏移3600 秒（一小时）。任务执行时间不变，执行的时候通过 `context.getDataTime()` 获取的是前一天23:30的数据。



结果验证

1. 在包含数据时间的任务创建完成后，进入**执行列表**页面，找到对应的任务，在**操作**列单击**详情**。
2. 在**任务实例详情**页面单击**基本信息**。
3. 在**基本信息**页签中确认任务的**数据时间**是否和设置的一致。

6.8.7. 如何重刷数据

通过重刷数据功能，您可以重新触发一段时间区间内的实例，来重刷业务数据。

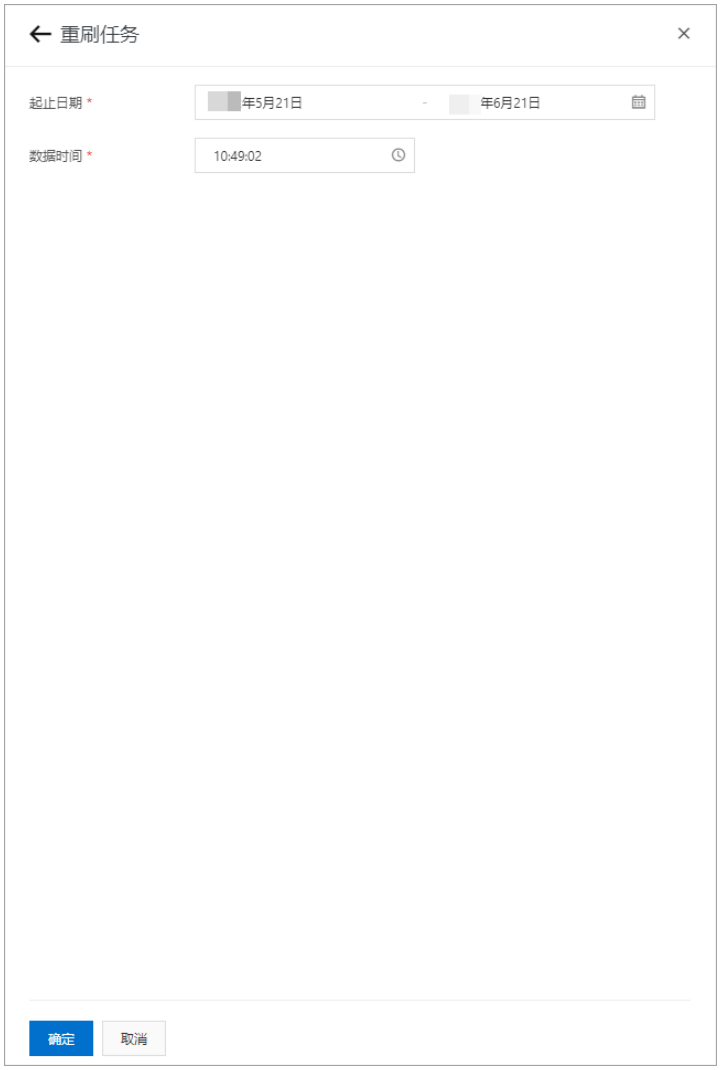
重刷调度任务

如果您的业务发生变更，如数据库增加一个字段或者上一个月数据有错误，需要把过去一段时间的任务重新执行一遍，可以重刷调度任务数据。

> **说明** 任务和工作流都支持重刷数据（只支持天级别的调度周期）。

如果您之前执行的某个调度任务的数据出现偏差或遗漏，您可以通过重新设置执行参数并执行某个调度任务属性、获取数据。

1. 在**任务管理**页面，单击目标任务**操作**列下的图标，然后单击**重刷任务**。
2. 在**重刷任务**面板，设置**起止日期**和**数据时间**，单击**确定**。
   - **起止日期**：指定重刷的日期区间。
   - **数据时间**：指定重刷日期区间内的重刷时间。



示例重刷配置如下：

- 当前时间为2019-01-01 10:00:00。
- 重刷任务的起止日期为2018-10-01~2018-10-07，默认从2018年10月1日00:00:00起，到2018年10月7日23:59:59结束。
- 数据时间为11:11:11。

则该任务会被重刷7次，生成7个实例。

序号	调度时间	数据时间
1	2019.1.1 10:00:00	2018.10.1 11:11:11
2	2019.1.1 10:00:00	2018.10.2 11:11:11
3	2019.1.1 10:00:00	2018.10.3 11:11:11

序号	调度时间	数据时间
4	2019.1.1 10:00:00	2018.10.4 11:11:11
5	2019.1.1 10:00:00	2018.10.5 11:11:11
6	2019.1.1 10:00:00	2018.10.6 11:11:11
7	2019.1.1 10:00:00	2018.10.7 11:11:11

6.9. 控制台使用指南

6.9.1. 执行列表

您可以通过执行列表查看当天任务执行情况，分为任务实例列表和流程实例列表。

查看执行列表

> **说明** 标准版中任务实例记录只支持查看10条，专业版支持查询100条任务实例记录。

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏单击**执行列表**。
5. 在**执行列表**页面的**所属微服务空间**列表选择具体的命名空间，然后单击**任务实例列表**或**流程实例列表**页签。

默认显示*全部*执行记录。
可以对执行记录进行筛选和搜索。

- 按*全部*、*成功*、*失败*、*运行*、*池子*和*等待*等状态进行筛选。
- 按**应用ID**、**任务ID**、**实例ID**等关键字进行搜索。
- 设置时间，按**分**、**小时天**、**周**、**月**、**自定义**时间维度进行筛选。

查看任务实例详情

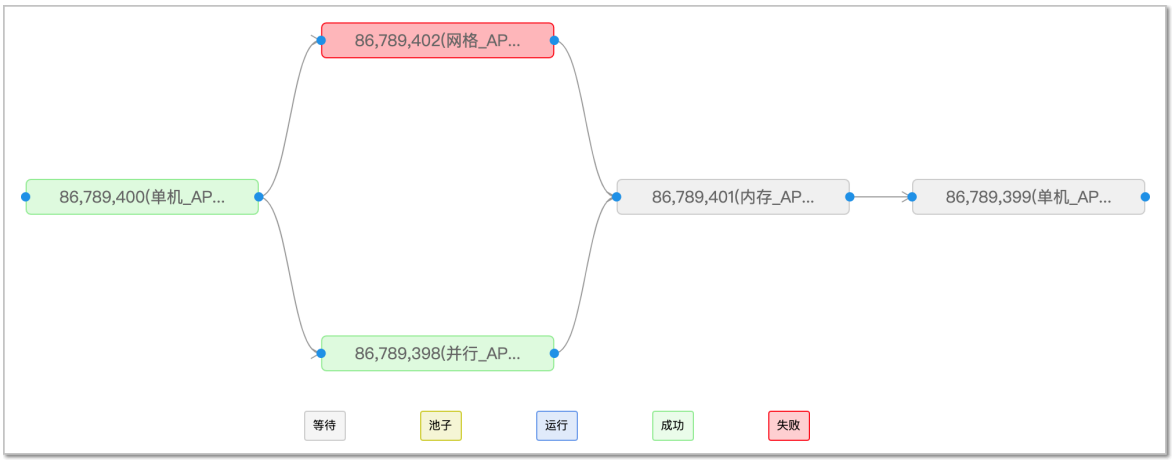
1. 在**执行列表**页面的**任务实例列表**页签找到您要查看的任务，然后在操作列单击**详情**。
2. 在**任务实例详情**页面的不同页签查看**基本信息**、**当前执行详情**、**子任务列表**、**执行日志**和**分片详情**。

   根据不同的执行方式，可以看到不同的执行详情。
   - **单机运行**：包含**基本信息**和**执行日志**
   - **广播运行**：包含**基本信息**和**当前执行详情**
   - **并行计算**：包含**基本信息**、**当前执行详情**和**子任务列表**
   - **内存网格**：包含**基本信息** 和**当前执行详情**
   - **网格计算**：包含**基本信息** 和**当前执行详情**
   - **分片运行**：包含**基本信息**和**分片详情**

3. 在**子任务列表**页签中，针对不同状态的子任务，可以进行不同操作。
   - **等待**：标记成功
   - **失败**：原地重跑、标记成功
   - **运行**：kill
   - **成功**：无

查看流程实例详情

1. 在**执行列表**页面的**流程实例列表**页签找到您要查看的流程，然后在**操作**列单击流程实例ID链接。
2. 在**流程画布**中查看该流程的详细信息。



各任务的颜色表示其执行状态。例如红色代表执行失败，则依赖它的任务只能处在等待状态（灰色）。

3. 右键单击执行失败的任务，在快捷菜单中单击**详情**查看失败原因。
4. 修改后，再右键单击该任务，在快捷菜单中单击**重跑**，重新执行。
   - 如果执行成功，下游任务会执行。
   - 如果不想再执行该任务，也可以在快捷菜单中单击**标记成功**，强制标记为成功状态，下游的任务也会开始执行。

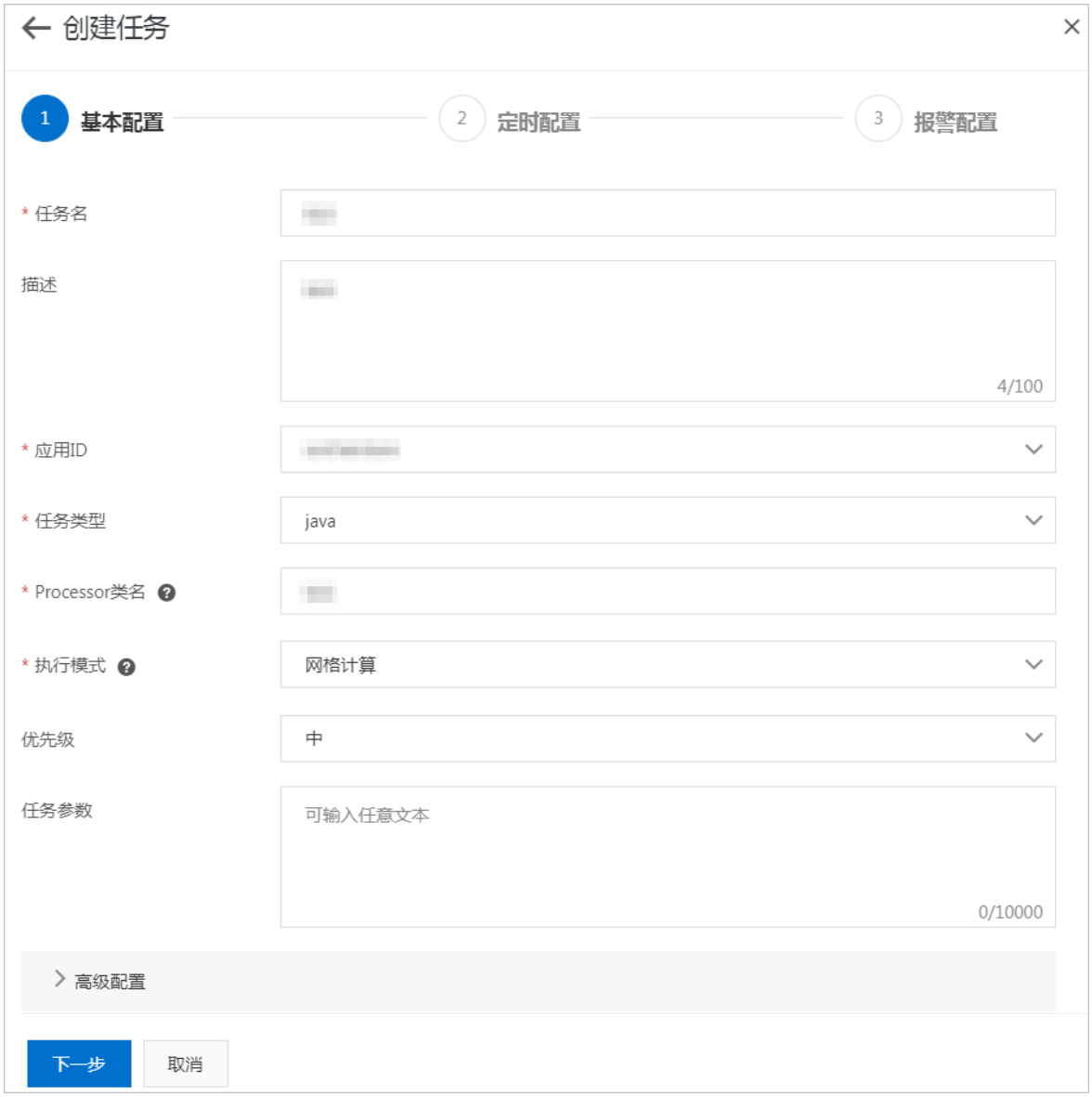
6.9.2. 任务管理

您可以在任务管理页面对调度任务进行一些列操作，包括创建、编辑、执行、复制、启禁用和删除，还可以重刷调度任务数据。

创建调度任务

> 注意　创建任务前，请确保您已经创建了任务分组。详情请参见应用管理。

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏选择**任务管理**。
5. 在**任务管理**页面，选择目标命名空间，单击**创建任务**。
6. 在**基本配置**配置向导页，设置调度任务的基本参数和高级配置参数，然后单击**下一步**。



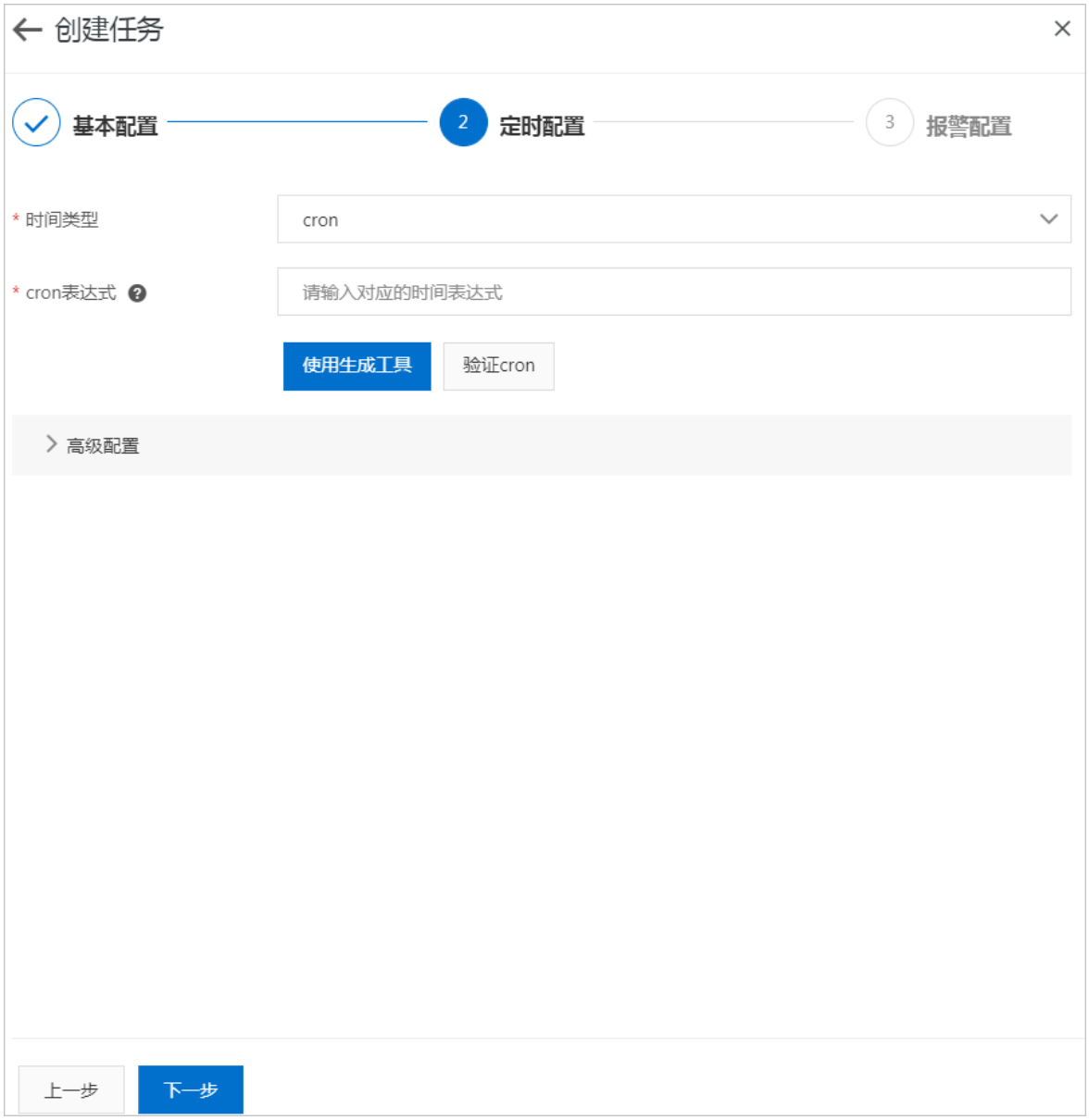
基本配置参数说明如下：

配置名称	意义
任务名	任务名称
描述	任务描述，尽量简洁地描述业务，便于后续搜索。
应用ID	任务所属分组。可以在下拉列表中选择。
任务类型	指任务所实现的语言，当前支持Java、Shell、Python、Go、http、Node.js、xxljob和DataWorks类型，其中Shell、Python和Go会弹出编辑框，在编辑框中编写任务脚本。
Processor类名（仅适用于Java任务类型）	JobProcessor的全路径，如xxx.xxx.xxx.HelloProcessor，仅任务类型选择Java时出现。

配置名称	意义
执行模式	执行模式，这里特指任务执行的模式，当前支持以下模式。 ○ 单机运行：随机选一台机器执行。 ○ 广播运行：所有机器同时执行并等待全部结束。 ○ 可视化MapReduce：Map模型，子任务300以下，有子任务列表。 专业版可支持至1,000以下，且支持业务关键字查询。 ○ 内存MapReduce：Map模型，子任务执行信息采用内存存储，速度快，子任务50,000以下，无子任务列表。 ○ 磁盘MapReduce：Map模型，子任务执行信息采用磁盘文件存储，吞吐量大，子任务1,000,000以下，无子任务列表。 ○ 分片运行：类似elastic-job模型，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。 **说明** 当选择了不同的执行模式后，高级设置中的参数会随之变化。
任务参数	任意字符串，可以在运行时通过上下文获取。

根据实际业务，如果需要进行高级配置需求，请参见任务管理高级配置参数说明进行配置。

7. 在**定时配置**配置向导页，设置定时参数和高级配置参数，然后单击**下一步**。



定时参数说明如下：

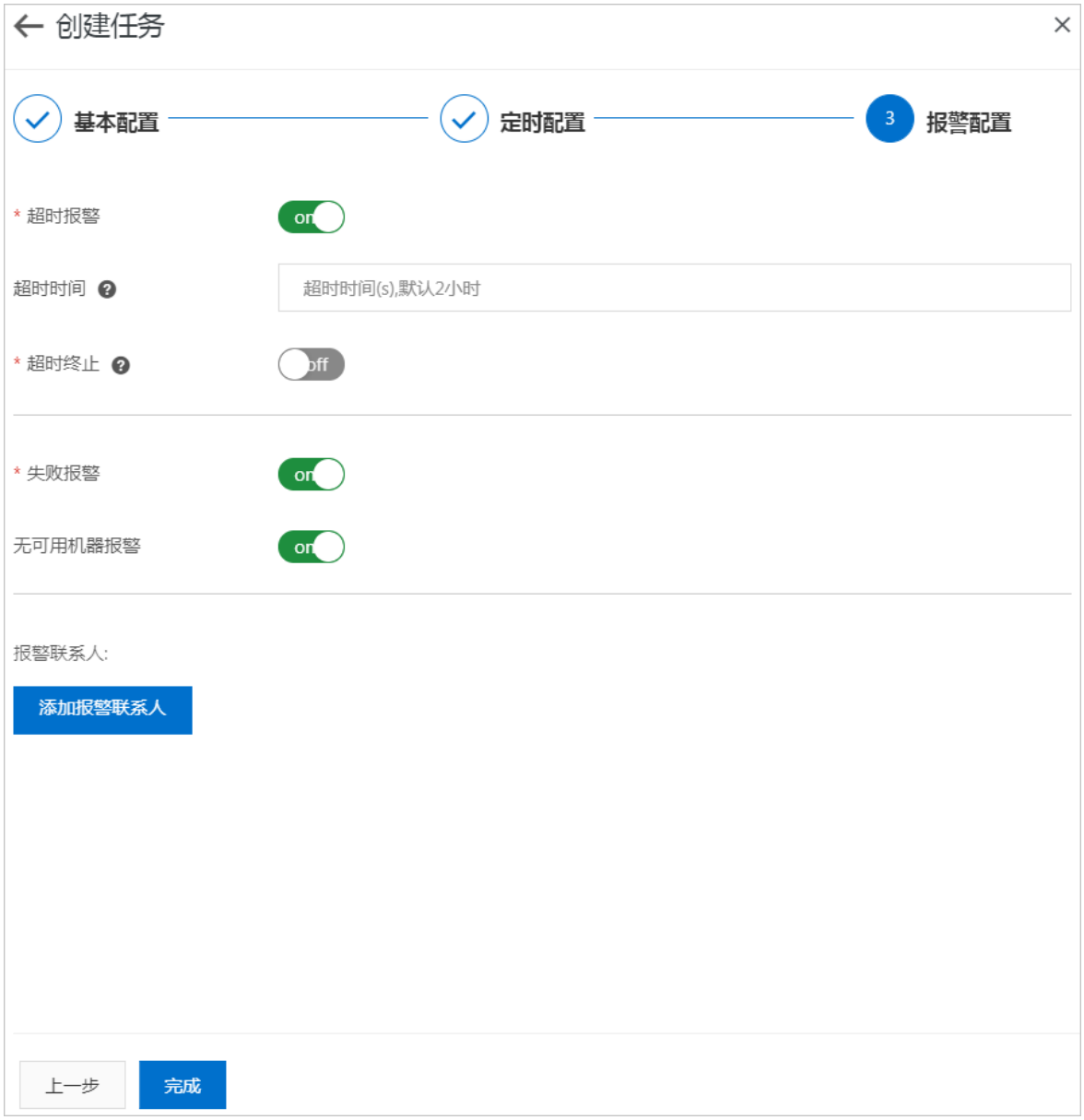
配置名称	意义
时间类型	◦ none：无调度方式，一般通过工作流触发。 ◦ cron：Cron表达式。 ◦ api：通过API触发。 ◦ fixed_rate：固定频率。 ◦ second_delay：秒级固定延迟。 ◦ onetime：一次性任务。
cron表达式（仅适用于cron时间类型）	填写Cron表达式。可以直接按照Cron语法填写，也可以使用工具生成并验证。
固定频率（仅适用于fixed_rate时间类型）	填写固定频率，单位为秒，只支持60秒以上。例如200表示每200s调度一次。

配置名称	意义
固定延迟（仅适用于second_delay时间类型）	填写固定延迟，单位为秒。范围为1秒~60秒。例如5表示延迟5秒触发调度

当时间类型选择Cron后，可以进行高级配置。高级配置参数说明如下：

配置名称	意义
时间偏移	数据时间相对于调度时间的偏移，可以在调度时从上下文获取该值。
时区	可以根据实际情况选择不同时区，包括一些常用国家或地区，也包括标准的GMT表达方式。

8. 在**报警配置**配置向导页，设置报警参数及联系人，然后单击**完成**。



9. 返回**任务管理**页面，查看刚才创建的任务是否已存在，且参数是否和配置一致。

> **说明** 调度任务创建完成后，默认为启用状态。您也可以根据实际情况禁用、再启用该任务。具体操作，请参见启用和禁用调度任务。

编辑调度任务

1. 在**任务管理**页面，单击目标任务的**操作**列下的**编辑**。
2. 在**编辑**面板，设置基本配置、定时配置和报警配置参数。

> **说明** 任务分组和任务类型在编辑时不可修改。其它参数配置规则和创建调度任务时一致。

执行调度任务

在**任务管理**页面任务列表的**操作**列下的**运行一次**，可以执行一次该调度任务。

启用和禁用调度任务

单个调度任务启用和禁用

1. 在**任务管理**页面，单击目标任务**操作**列下的图标，然后单击**禁用**或**启用**。
2. 在确认对话框单击**确认**。

批量启用和禁用调度任务

如果是专业版应用，您可以批量启用和禁用调度任务。在**任务管理**页面，勾选目标任务，然后单击任务列表下方的**批量禁用**或**批量启用**。

复制调度任务

在**任务管理**页面，单击目标任务**操作**列下的图标，然后单击**复制**。可以复制该调度任务的配置，您可以编辑复制的任务，生成新的调度任务。

重刷调度任务

如果您的业务发生变更，如数据库增加一个字段或者上一个月数据有错误，需要把过去一段时间的任务重新执行一遍，可以重刷调度任务数据。

> **说明** 任务和工作流都支持重刷数据（只支持天级别的调度周期）。

如果您之前执行的某个调度任务的数据出现偏差或遗漏，您可以通过重新设置执行参数并执行某个调度任务属性、获取数据。

1. 在**任务管理**页面，单击目标任务**操作**列下的图标，然后单击**重刷任务**。
2. 在**重刷任务**面板，设置**起止日期**和**数据时间**，单击**确定**。
   - **起止日期**：指定重刷的日期区间。
   - **数据时间**：指定重刷日期区间内的重刷时间。

   示例重刷配置如下：
   - 当前时间为2019-01-01 10:00:00。
   - 重刷任务的起止日期为2018-10-01~2018-10-07，默认从2018年10月1日00:00:00起，到2018年10月7日23:59:59结束。
   - 数据时间为11:11:11。

   则该任务会被重刷7次，生成7个实例。

序号	调度时间	数据时间
1	2019.1.1 10:00:00	2018.10.1 11:11:11
2	2019.1.1 10:00:00	2018.10.2 11:11:11
3	2019.1.1 10:00:00	2018.10.3 11:11:11
4	2019.1.1 10:00:00	2018.10.4 11:11:11
5	2019.1.1 10:00:00	2018.10.5 11:11:11
6	2019.1.1 10:00:00	2018.10.6 11:11:11
7	2019.1.1 10:00:00	2018.10.7 11:11:11

查看调度任务的执行记录和操作记录

- 查看执行记录：在**任务管理**页面，单击目标任务**操作**列下的图标，单击**历史记录**，查看该调度任务的执行记录。
- 查看操作记录：在**任务管理**页面，单目标任务**操作**列下的图标，单击**操作记录**，查看该调度任务的管理操作记录。

删除调度任务

1. 在**任务管理**页面，单击目标任务**操作**列下的图标，单击**删除**。
2. 在弹出的确认对话框中单击**确认**。

6.9.3. 流程管理

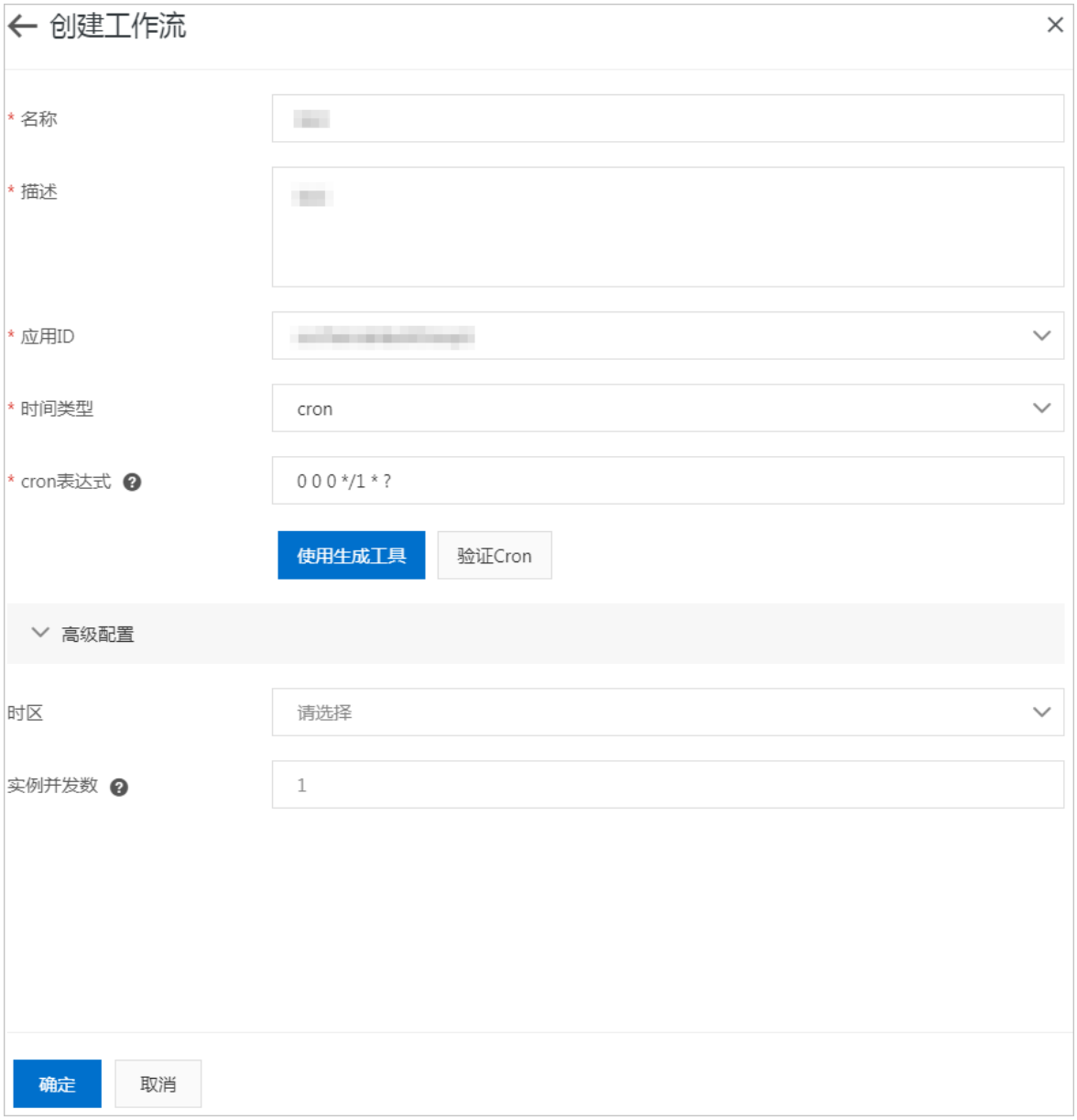
流程管理提供可视化的任务编排，您可以使用Crontab创建定时调度工作流，并通过API触发。

创建工作流

您可以创建工作流调度任务。

> **说明** 目前工作流调度仅支持Cron表达式。

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏单击**流程管理**。
5. 在**流程管理**页面选择目标命名空间，然后单击**创建工作流**。
6. 在**创建工作流**面板，设置工作流的**名称**、**描述**、**应用ID**和**时间类型**（包括Cron和API），然后单击**确定**。



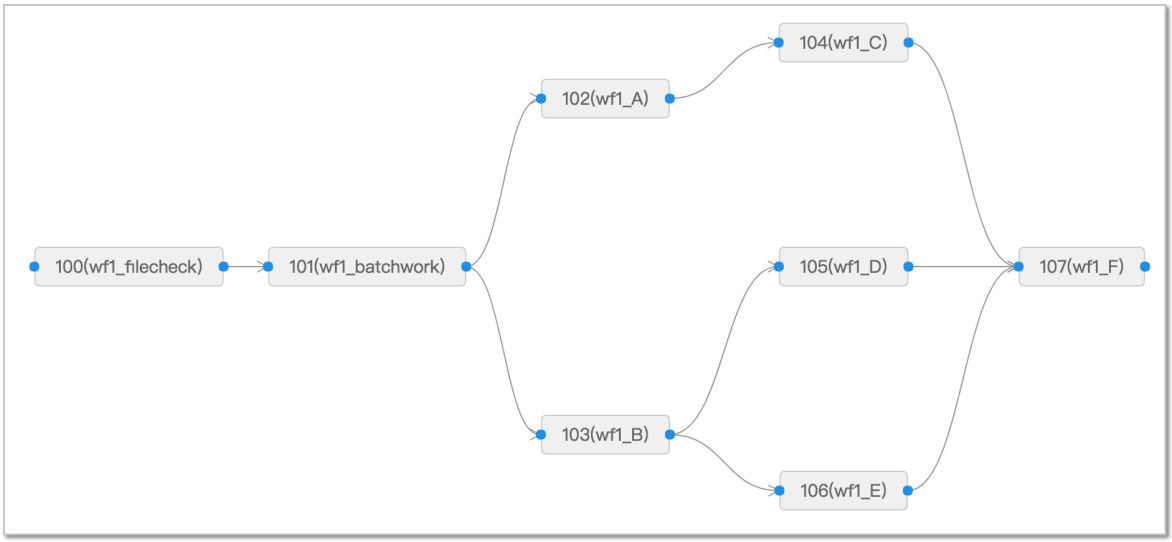
也可以单击**高级设置**，设置**时区**和**实例并发数**。

7. 在**工作流详情**页面，单击**创建任务**或**导入任务**，添加调度任务。
   - **创建任务**：和创建调度任务的步骤一致，请参见创建调度任务。
   - **导入任务**：将已创建的Job导入到工作流中。

   > **说明** 导入Job会弹出 `导入Job将会移除该Job的定时配置` 提示框，单击**确认**，则该Job不会独立调度，会跟随工作流的调度周期进行调度。

8. 调度任务添加完毕后，按住并拖动任务两侧的端点到其它任务的端点连接调度任务，形成依赖关系，组成完成的工作流。

   在工作流中，上下游的任务还可以实现数据传递。
   如果需要删除某两个任务间的依赖关系，选中它们之间的线条，单击**Delete**；右键某个调度任务，在快捷菜单中单击**删除**，即可删除该任务。
   一个工作流示意如下：



在该工作流中，101执行完，102和103会同时开始执行。104，105，106都执行成功，107才会开始执行。

9. 工作流配置完成后，单击**发布**。

   工作流创建完成后，返回**流程管理**页面，可以查看是否已经包含创建的流程及相关信息。

后续操作

工作流发布之后，就会变成启用状态并自动开始调度。如果不想马上启用，可以返回**流程管理**页面，在**操作**列单击**运行一次**测试下，没问题再手动启用。

您还可以对工作流进行编辑、查看操作记录和历史记录，以及删除、重刷任务等操作。

6.9.4. 应用管理

您可以通过应用管理页面创建应用（任务分组）、测试该应用的连接机器、为RAM用户进行分组授权等操作，如果不需要该应用，还可以删除应用。

创建应用

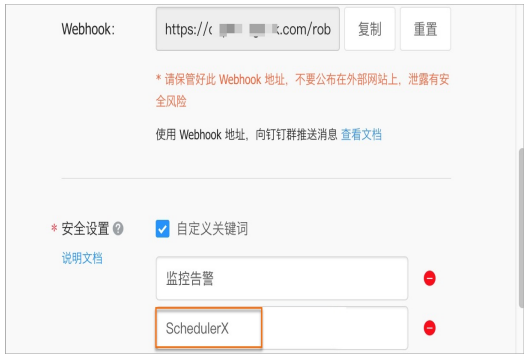
在创建调度任务前，您需要先创建应用。

1. 在顶部菜单栏选择地域。
2. 在**应用管理**页面选择目标命名空间，然后单击**创建应用**。
3. 在**基本配置**向导页，输入**应用名**、**应用ID**并且选择应用类型，设置高级配置参数（可选），然后单击**下一步**。

参数	解释	默认值
应用名	自定义设置应用名称。	无
应用ID	**应用ID**为应用接入的GroupID，保证同一个命名空间下唯一，否则将创建失败，可以和**应用名**一致。	无
load5	不能超过客户端机器CPU可用核数	0
应用类型	◦ 普通应用：非K8s部署的应用，或者对K8s任务没有需求。 ◦ K8s应用：应用通过K8s部署，并且有需求要使用K8s任务。	普通应用
内存使用率	表示近5分钟进程内存平均使用率不能大于该阈值，否则判断客户端机器繁忙。	90%
磁盘使用率	表示磁盘使用率不能大于该值，否则判断客户端机器不健康，状态繁忙。	95%
是否触发繁忙机器	机器繁忙时是否继续触发客户端执行。	打开
高级配置		
任务最大数量	一个分组最多支持的Job数量。	1000
自动扩容	选择是否自动扩容。开启时，需要设置**全局任务数**。	关闭

参数	解释	默认值
流控	选择是否流控。开启时，需要设置**任务实例并发数**。	关闭
版本	根据需求选择版本，版本对比请参见产品版本对比。	专业版
日志服务	开启后，增加一个Log4j或Logback的配置，即可在控制台看到每次任务调度（包括分布式任务）的业务日志，方便排查问题。	关闭

4. 在**报警配置**向导页填写相关信息，选择**报警渠道**并设置**报警联系人**。报警方式支持报警联系人组、自定义两种方式。
   - 报警联系人组
     报警联系组下所有联系人都可以接收到报警通知。关于创建报警联系人组，请参见创建报警联系人或报警联系组。
   - 自定义
     单独添加报警联系人。需要设置多个报警联系人时，单击**添加报警联系人**，然后输入联系人信息。

参数	描述
报警渠道	目前支持**短信**、**邮件**、**Webhook**和**电话**。
昵称	自定义设置报警联系人的昵称。
邮件	输入报警联系人的真实邮件地址。
Webhook	◦ 目前支持企业微信、飞书、钉钉三种。且可以同时配置多个Webhook机器人，机器人链接用英文逗号（,）分割。 ◦ 需要在钉钉机器人增加关键字“SchedulerX”（注意大小写），否则会收不到告警信息。  ◦ 获取Webhook的方法请参见钉钉开发文档、企业微信开发文档和飞书开发文档。
手机号	输入报警联系人的真实手机号码。

应用创建成功后，应用列表页会自动刷新。

> **注意**
> - 应用ID：客户端初始化需要填写的参数，也是一个分组的唯一标识。
> - 应用Key：SDK请求需要填写，用来进行请求验证具体可以参考SDK文档，请勿外传。

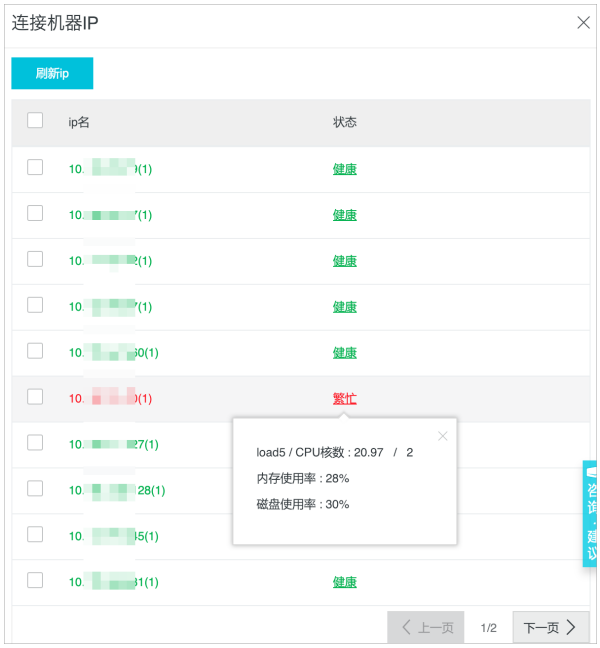
查看实例

当您的应用接入任务调度并部署到EDAS后，您可以通过查看实例检查该应用的调度任务是否接入成功。操作步骤如下：

1. 在**应用管理**页面分组列表的**实例总数**列查看实例数量。

   > **说明** 如果**实例总数**为0，则说明该应用的任务调度接入失败。

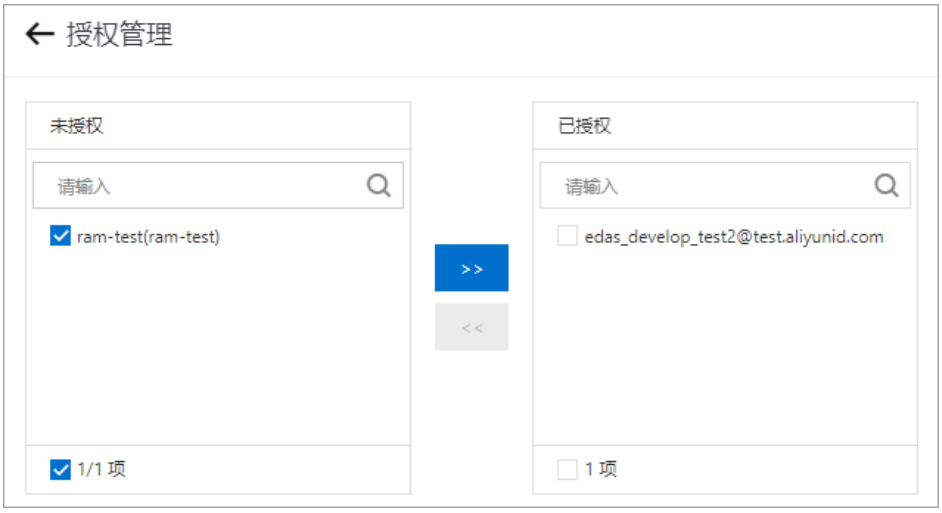
2. 在**操作**列单击**查看实例**。
3. 在**查看实例**页面查看该应用下的客户端机器列表以及每台客户端机器的状态。
   - 客户端机器IP后面括号里的数字表示该机器上有多少SchedulerX客户端。一般情况都是1个，但支持一台机器起多个客户端。



   - 客户端机器包含健康和繁忙两种状态。默认不会再向繁忙状态的机器下发任务。如果出现极端情况，如所有的连接机器都是繁忙状态，无可用机器将造成任务下发失败。如果想向繁忙机器强制下发任务。可以在告警配置中，将**过滤忙碌机器**开关设置为关闭。

授权

当前支持应用级别授权，主账号和子账号可以授权给该主账号下其他子账号。



如果点击授权的时候系统提示“没有权限”，您需要用主账号给子用户添加一个自定义的权限策略，权限策略内容如下：

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

删除应用

在您确认不会再通过某个应用创建调度任务后，可以删除应用。应用删除后，该分组关联的所有调度任务将不能再执行。

1. 在**应用管理**页面，单击目标应用的**操作**列下的**删除**。
2. 在**删除应用**对话框，确认应用信息并手动输入应用名称，然后单击**确定**。

> **说明**
> - 删除分组后，分组将从列表中去除。
> - 删除分组后，分组所有关联的任务将不触发，同时页面不显示。
> - 如果误删除分组，您需要提交工单进行恢复。

6.9.5. 操作记录

您可以通过操作记录页面，查看应用分组、调度任务和流程的操作记录。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏单击**任务调度**。
3. 在顶部菜单栏选择地域。
4. 在左侧导航栏单击**操作记录**。
5. 在**操作记录**页面选择目标命名空间，查看操作记录。

6.10. API参考

6.10.1. API概览

分布式任务调度SchedulerX2.0提供以下API接口。

API列表

API	描述
创建应用分组	调用CreateAppGroup创建应用分组。
CreateJob	调用CreateJob创建任务。
GetJobInfo	调用GetJobInfo获取指定Jobid任务详情，通常用来更新任务。
UpdateJob	调用UpdateJob更新任务配置信息。
DeleteJob	调用DeleteJob删除指定任务。
BatchDeleteJobs	调用BatchDeleteJobs批量删除任务。
EnableJob	调用EnableJob启用指定任务。
BatchEnableJobs	调用BatchEnableJobs批量启用任务。
DisableJob	调用DisableJob停用指定任务。
BatchDisableJobs	调用BatchDisableJobs批量禁用任务。
ExecuteJob	调用ExecuteJob触发一次任务。
StopInstance	调用StopInstance终止某次正在运行的实例。
GetJobInstanceList	调用GetJobInstanceList获取指定任务ID的执行实例列表。
GetJobInstance	调用GetJobInstance获取指定任务实例详情。
ExecuteWorkflow	调用ExecuteWorkflow触发一次工作流。
EnableWorkflow	调用EnableWorkflow启用指定工作流。
DisableWorkflow	调用DisableWorkflow禁用指定工作流。
DeleteWorkflow	调用DeleteWorkflow删除指定工作流。
ListJobs	调用ListJobs获取任务列表。
ListGroups	调用ListGroups获取应用列表。
ListNamespaces	调用ListNamespaces接口获取命名空间列表。

6.10.2. 调用方式

您可以使用SDK调用分布式任务调度2.0的API，完成日常的管理操作。

RAM用户授权

您可以使用RAM用户账号调用分布式任务调度2.0的API，但需要先获取、配置账号的AccessKey ID和AccessKey Secret，并由阿里云账号对RAM用户账号授予相应资源的操作权限。

支持的地域列表

在使用SDK调用API的时候需要用到地域的Region ID、Domain等信息。

> **说明** Domain（VPC）适用于没有公网的VPC的环境。

名称	RegionId	Domain	Domain（VPC）
华东1（杭州）	cn-hangzhou	schedulerx.cn-hangzhou.aliyuncs.com	schedulerx-vpc.cn-hangzhou.aliyuncs.com
华东2（金融云）	cn-shanghai-finance-1	schedulerx.cn-shanghai-finance-1.aliyuncs.com	schedulerx-vpc.cn-shanghai-finance-1.aliyuncs.com
华南1（深圳）	cn-shenzhen	schedulerx.cn-shenzhen.aliyuncs.com	schedulerx-vpc.cn-shenzhen.aliyuncs.com
华东2（上海）	cn-shanghai	schedulerx.cn-shanghai.aliyuncs.com	schedulerx-vpc.cn-shanghai.aliyuncs.com
华北2（北京）	cn-beijing	schedulerx.cn-beijing.aliyuncs.com	schedulerx-vpc.cn-beijing.aliyuncs.com
华北3（张家口）	cn-zhangjiakou	schedulerx.cn-zhangjiakou.aliyuncs.com	schedulerx-vpc.cn-zhangjiakou.aliyuncs.com
中国（香港）	cn-hongkong	schedulerx.cn-hongkong.aliyuncs.com	schedulerx-vpc.cn-hongkong.aliyuncs.com
新加坡	ap-southeast-1	schedulerx.ap-southeast-1.aliyuncs.com	schedulerx-vpc.ap-southeast-1.aliyuncs.com
美国（弗尼吉亚）	us-east-1	schedulerx.us-east-1.aliyuncs.com	schedulerx-vpc.us-east-1.aliyuncs.com
德国（法兰克福）	eu-central-1	schedulerx.eu-central-1.aliyuncs.com	schedulerx-vpc.eu-central-1.aliyuncs.com
日本（东京）	ap-northeast-1	schedulerx.ap-northeast-1.aliyuncs.com	schedulerx-vpc.ap-northeast-1.aliyuncs.com
澳大利亚（悉尼）	ap-southeast-2	schedulerx.ap-southeast-2.aliyuncs.com	schedulerx-vpc.ap-southeast-2.aliyuncs.com
政务云	cn-north-2-gov-1	schedulerx.cn-north-2-gov-1.aliyuncs.com	schedulerx-vpc.cn-north-2-gov-1.aliyuncs.com
公网测试	public	schedulerx.aliyuncs.com	无

添加SDK依赖

在应用程序的*pom.xml*文件中添加OpenAPI的SDK依赖，SDK最新版本，请参见Aliyun Java SDK Schedulerx2。

使用SDK调用API

下面以一个通用的示例说明如何使用SDK调用API。

```
public static void main(String[] args) throws Exception{
     // OpenAPI的接入点，具体参见支持地域列表和购买实例的地域。
     String regionId = "XXXXX";
     //鉴权使用的AccessKey ID。
     String accessKeyId = "XXXXXXXXXXXXXXXXX";
     //鉴权使用的AccessKey Secret。
     String accessKeySecret = "XXXXXXXXXXXXXXXXX";
     //产品名称
     String productName ="schedulerx2";
     //参见支持的地域列表，选择Domain。
     String domain ="schedulerx.xxxx.aliyuncs.com";
     //构建OpenAPI客户端。
     DefaultProfile.addEndpoint(regionId, productName, domain);
     DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySecre
t);
     DefaultAcsClient defaultAcsClient = new DefaultAcsClient(defaultProfile);
     //构建接口请求（根据接口中参数类型和是否必填进行填充）。
     EnableJobRequest request = new EnableJobRequest();
     request.setJobId(jobId);
     request.setNamespace("xxxxxx");
     request.setGroupId("yyyyyy");
     //发送请求。
     EnableJobResponse response = defaultAcsClient.getAcsResponse(request);
     if (!response.getSuccess()) {
         System.out.println(response.getMessage());
         System.out.println("EnableJob: "+response.getRequestId());
     }
}
```

6.10.3. 获取AccessKey

您可以为阿里云账号和RAM用户创建一个访问密钥（AccessKey）。在调用阿里云API时您需要使用AccessKey完成身份验证。

背景信息

AccessKey包括AccessKey ID和AccessKey Secret。

- AccessKey ID：用于标识用户。
- AccessKey Secret：用于验证用户的密钥。AccessKey Secret必须保密。

> 警告　阿里云账号AccessKey泄露会威胁您所有资源的安全。建议使用RAM用户AccessKey进行操作，可以有效降低AccessKey泄露的风险。

操作步骤

1. 使用阿里云账号登录阿里云管理控制台。
2. 将鼠标置于页面右上方的账号图标，单击**AccessKey 管理**。
3. 在**安全提示**页面，选择获取阿里云账号或者RAM用户的Accesskey。

4. 获取账号AccessKey。
   - 获取阿里云账号AccessKey
     a. 在**安全提示**页面单击**继续使用AccessKey**。
     b. 在**安全信息管理**页面，单击**创建AccessKey**。
     c. 在**手机验证**页面，获取并填入**校验码**，单击**确定**。
     d. 在**新建用户AccessKey**页面，展开**AccessKey详情**，查看**AccessKey ID**和**AccessKey Secret**。可以单击**保存AK信息**，下载AccessKey信息。

   - 获取RAM用户AccessKey
     a. 在**安全提示**页面单击**开始使用子用户AccessKey**
        - 如果是未创建RAM用户，请在系统跳转的RAM访问控制台的**创建用户**页面，创建RAM用户，创建完成后会自动为RAM用户生成访问密钥（AccessKey）。详情请参见创建RAM用户。
        - 如果是已创建RAM用户，继续执行后续步骤直接获取RAM用户的AccessKey。
     b. 登录RAM访问控制台，在左侧导航栏选择**人员管理 > 用户**。
     c. 在**用户**页面搜索需要获取AccessKey的用户。
     d. 单击目标用户登录名称，在用户详情页**认证管理**页签下的**用户 AccessKey**区域，单击**创建AccessKey**。

e. 在**创建 AccessKey**页面，查看**AccessKey ID**和**AccessKey Secret**。
您还可以单击**下载CSV文件**或者**复制**，完成对AccessKey信息的保存。

6.10.4. BatchDeleteJobs

调用BatchDeleteJobs批量删除任务。

> **说明** 调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
    <version>1.0.4</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	BatchDeleteJobs	系统规定参数。取值：BatchDeleteJobs。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的**应用管理**页面中获取。
JobIdList.N	RepeatList	是	99341	任务ID列表，多个任务ID以半角逗号（,）分隔。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	任务所属命名空间ID，在控制台**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	任务所属地域。

名称	类型	是否必选	示例值	描述
NamespaceSource	String	否	Schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	状态码
Message	String	message	附加信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7	请求唯一ID
Success	Boolean	true	批量删除任务是否成功。取值如下： • **true**：成功 • **false**：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=BatchDeleteJobs
&GroupId=testSchedulerx.defaultGroup
&JobIdList.1=99341
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<BatchDeleteJobsResponse>
  <Message>delete failed jobs=[675946,676129]</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7</RequestId>
  <Code>200</Code>
  <Success>true</Success>
</BatchDeleteJobsResponse>
```

`JSON` 格式

```
{
    "Message": "delete failed jobs=[675946,676129]",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7",
    "Code": 200,
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import com.google.common.collect.Lists;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.BatchDeleteJobsRequest;
import com.aliyuncs.schedulerx2.model.v20190430.BatchDeleteJobsResponse;
public class TestBatchDeleteJobs {
    public static void main(String[] args) throws Exception {
        // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写.
        String regionId = "cn-hangzhou";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "xxxxxx";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "xxxxxxxx";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-hangzhou.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        BatchDeleteJobsRequest request = new BatchDeleteJobsRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobIdLists(Lists.newArrayList(3982L,3984L));
        BatchDeleteJobsResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("BatchDeleteJob: "+response.getRequestId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

6.10.5. BatchDisableJobs

调用BatchDisableJobs接口批量禁用任务。

说明 在调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
    <version>1.0.4</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	BatchDisableJobs	系统规定参数。取值：BatchDisableJobs。
JobIdList.N	RepeatList	是	99341	任务ID列表，多个任务ID以半角逗号（,）分隔。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	任务所属命名空间ID，在控制台**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	任务所属地域。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的**应用管理**页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	状态码
Message	String	disable failed jobs=[99341]	附加信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7	请求唯一ID
Success	Boolean	true	批量禁用任务是否成功。取值如下： • **true**：成功 • **false**：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=BatchDisableJobs
&JobIdList.1=99341
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<BatchDisableJobsResponse>
  <Message>disable failed jobs=[99341]</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7</RequestId>
  <Code>200</Code>
  <Success>true</Success>
</BatchDisableJobsResponse>
```

`JSON` 格式

```
{
    "Message": "disable failed jobs=[99341]",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7",
    "Code": 200,
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import com.google.common.collect.Lists;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.BatchDisableJobsRequest;
import com.aliyuncs.schedulerx2.model.v20190430.BatchDisableJobsResponse;
public class TestBatchDisableJobs {
    public static void main(String[] args) throws Exception {
        // OpenAPI 的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-hangzhou";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "xxxxxx";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "xxxxxxxx";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-hangzhou.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        BatchDisableJobsRequest request = new BatchDisableJobsRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobIdLists(Lists.newArrayList(3982L,3984L));
        BatchDisableJobsResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("BatchDisableJob: "+response.getRequestId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

6.10.6. BatchEnableJobs

调用BatchEnableJobs批量启用任务。

说明 在调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
    <version>1.0.4</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	BatchEnableJobs	系统规定参数。取值：BatchEnableJobs。
JobIdList.N	RepeatList	是	99341	任务ID列表，多个任务ID以半角逗号（,）分隔。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	任务所属命名空间ID，在控制台**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	任务所属地域。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的**应用管理**页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	状态码
Message	String	message	附加信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7	请求唯一ID
Success	Boolean	true	批量启用任务是否成功。取值如下： • **true**：成功 • **false**：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=BatchEnableJobs
&JobIdList.1=99341
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<BatchEnableJobsResponse>
  <Message>enable failed jobs=[99342]</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7</RequestId>
  <Code>200</Code>
  <Success>true</Success>
</BatchEnableJobsResponse>
```

`JSON` 格式

```
{
    "Message": "enable failed jobs=[99342]",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB5A7",
    "Code": 200,
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import com.google.common.collect.Lists;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.BatchEnableJobsRequest;
import com.aliyuncs.schedulerx2.model.v20190430.BatchEnableJobsResponse;
public class TestBatchEnableJobs {
    public static void main(String[] args) throws Exception {
        // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-hangzhou";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "xxxxxx";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "xxxxxxxx";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-hangzhou.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        BatchEnableJobsRequest request = new BatchEnableJobsRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobIdLists(Lists.newArrayList(3982L,3984L));
        BatchEnableJobsResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("BatchEnableJob: "+response.getRequestId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

6.10.7. CreateAppGroup

调用CreateAppGroup创建应用分组。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
AppName	String	是	DocTest	应用名称。
GroupId	String	是	TestSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。

名称	类型	是否必选	示例值	描述
NamespaceName	String	是	Test	命名空间名称。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
Description	String	否	Test	应用描述。
MaxJobs	Integer	否	1000	最大任务数。
AlarmJson	String	否	AlarmTest	告警JSON。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Data	Struct		任务分组信息
AppGroupId	Long	6607	任务分组ID
Message	String	Your request is denied as lack of ssl protect.	错误信息，仅错误时返回错误信息。
RequestId	String	883AFE93-FB03-4FA9-A958-E750C6DE120C	请求唯一ID
Success	Boolean	true	创建应用是否成功。取值如下： • **true**：创建应用成功。 • **false**：创建应用失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateAppGroup
&AppName=DocTest
&GroupId=TestSchedulerx.defaultGroup
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&NamespaceName=Test
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<CreateAppGroupResponse>
  <RequestId>883AFE93-FB03-4FA9-A958-E750C6DE120C</RequestId>
  <Message>Your request is denied as lack of ssl protect.</Message>
  <Data>
        <AppGroupId>6607</AppGroupId>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</CreateAppGroupResponse>
```

`JSON` 格式

```
{
    "RequestId": "883AFE93-FB03-4FA9-A958-E750C6DE120C",
    "Message": "Your request is denied as lack of ssl protect.",
    "Data": {
        "AppGroupId": 6607
    },
    "Code": 200,
    "Success": true
}
```

示例Demo

```
package com.alibaba.schedulerx.pop;
import com.alibaba.schedulerx.common.util.JsonUtil;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.CreateAppGroupRequest;
import com.aliyuncs.schedulerx2.model.v20190430.CreateAppGroupResponse;
public class TestCreateAppGroup {
    public static void main(String[] args) throws Exception {
        // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的AccessKeyId，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的AccessKeySecret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        CreateAppGroupRequest request = new CreateAppGroupRequest();
        request.setNamespace("xxxxxx");
        request.setNamespaceName("xxxx");
        request.setAppName("xxxx");
        request.setDescription("xxx");
        request.setGroupId("xxx");
        //发送请求。
        CreateAppGroupResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("createApp: "+response.getRequestId());
        } else {
            System.out.println(JsonUtil.toJson(response));
        }
    }
}
```

6.10.8. CreateJob

调用CreateJob创建任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateJob	系统规定参数。取值：CreateJob。

名称	类型	是否必选	示例值	描述
ExecuteMode	String	是	standalone	任务执行模式，目前支持以下几种任务执行模式： • **单机运行**：standalone • **广播运行**：broadcast • **并行计算**：parallel • **内存网格**：grid • **网格计算**：batch • **分片运行**：sharding
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
JobType	String	是	Java	任务类型，目前支持以下几种任务类型： - **java** - **python** - **shell** - **go** - **http** - **nodejs**
Name	String	是	helloworld	任务名。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
TimeType	Integer	是	1	时间类型，目前支持以下几种时间类型： • **cron**：1 • **fixed_rate**：3 • **second_delay**：4 • **api**：100
Description	String	否	Test	任务描述。
Content	String	否	echo 'hello'	任务类型选择为**python**、**shell**、**go**时，必填的脚本代码内容。
Parameters	String	否	test	用户自定义参数，运行时可以获取。
MaxConcurrency	Integer	否	1	最大同时运行实例数量，默认值为1，即上次触发没有运行结束，即使到了运行时刻也不会进行下次触发。
MaxAttempt	Integer	否	0	错误最大重试次数，根据业务需求填写，默认值为0。
AttemptInterval	Integer	否	30	错误重试间隔，单位s，默认值为30。

名称	类型	是否必选	示例值	描述
ClassName	String	否	com.alibaba.schedulerx.test.helloworld	任务接口类完整路径。 当您选择Java任务类型时，才有该字段且必须填写完整路径。
JarUrl	String	否	暂不支持，不用填写	上传到OSS的完整路径。 如果选择JAR包运行，可以将相应JAR包上传到OSS的该路径下。
PageSize	Integer	否	100	并行网格任务高级配置，单次拉取子任务数，默认值为100。
ConsumerSize	Integer	否	5	并行网格任务高级配置，单机单次触发执行线程数，默认值为5。
QueueSize	Integer	否	10000	并行网格任务高级配置，子任务队列缓存上限，默认值为10000。
DispatcherSize	Integer	否	5	并行网格任务高级配置，子任务分发线程数，默认值为5。
TimeExpression	String	否	0 0/10 * * * ?	时间表达式，根据选择的时间类型设置时间表达式。 • **cron**：填写标准的cron表达式，支持在线验证。 • **api**：无时间表达式。 • **fixed_rate**：填写具体固定频率值，单位s。如30表示每隔30s触发一次。 • **second_delay**：填写固定延迟多少秒执行一次（1s~60s可选）。
Calendar	String	否	暂不支持，不用填写	cron类型可以选择填写自定义日历。
DataOffset	Integer	否	2400	cron类型可以选择时间偏移，单位s。
TimeoutEnable	Boolean	否	false	超时报警开关。取值如下： • **true**：开启超时报警开关。 • **false**：关闭超时报警开关。
Timeout	Long	否	7200	超时阈值，单位s，默认值7200。
TimeoutKillEnable	Boolean	否	false	超时终止开关。取值如下： • **true**：开启超时终止开关。 • **false**：关闭超时终止开关。
FailEnable	Boolean	否	false	失败报警开关。取值如下： • **true**：开启失败报警开关。 • **false**：关闭失败报警开关。
SendChannel	String	否	sms	报警发送形式，目前只支持短信发送报警，默认值sms。
ContactInfo.N.UserName	String	否	userA	报警联系人姓名。

名称	类型	是否必选	示例值	描述
ContactInfo.N.UserPhone	String	否	1381111****	报警接收手机号。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
TaskMaxAttempt	Integer	否	0	并行网格任务高级配置，子任务失败重试次数，默认值为0。
TaskAttemptInterval	Integer	否	0	并行网格任务高级配置，子任务失败重试间隔，默认值为0。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Data	Struct		任务详细信息
JobId	Long	92583	任务ID
Message	String	groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	错误信息，仅错误时返回错误信息。
RequestId	String	39090022-1F3B-4797-8518-6B61095F1AF0	请求唯一ID
Success	Boolean	true	创建任务是否成功。取值如下： • **true**：创建任务成功。 • **false**：创建任务失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateJob
&ExecuteMode=standalone
&GroupId=testSchedulerx.defaultGroup
&JobType=java
&Name=helloworld
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&TimeType=1
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<CreateJobResponse>
  <RequestId>39090022-1F3B-4797-8518-6B61095F1AF0</RequestId>
  <Message>groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9-bbaa
-20e90ffc****</Message>
  <Data>
        <JobId>92583</JobId>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</CreateJobResponse>
```

`JSON` 格式

```
{
    "RequestId": "39090022-1F3B-4797-8518-6B61095F1AF0",
    "Message": "groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9
-bbaa-20e90ffc****",
    "Data": {
        "JobId": 92583
    },
    "Code": 200,
    "Success": true
}
```

创建Java任务

```
package com.alibaba.schedulerx.pop;
import java.util.ArrayList;
import java.util.List;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobRequest.ContactInfo;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobResponse;
public class CreateJavaJob {
    public static void main(String[] args) throws Exception {
     // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的AccessKeyId，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的AccessKeySecret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        CreateJobRequest request = new CreateJobRequest();
        request.setJobType("java");
        request.setExecuteMode("standalone");
        request.setDescription("test");
        request.setName("helloworld");
        request.setClassName("com.alibaba.schedulerx.test.helloworld");
        request.setTimeType(1);
```

```
        request.setTimeExpression("0 0/10 * * * ?");
        request.setNamespace("xxxxx");
        request.setGroupId("xxxxxxx");
        // monitor
        request.setTimeoutEnable(true);
        request.setTimeoutKillEnable(true);
        request.setFailEnable(true);
        request.setTimeout(12300L);
        List<ContactInfo> contactInfosList = new ArrayList<>();
        ContactInfo contactInfo1 = new ContactInfo();
        contactInfo1.setUserName("userA");
        contactInfo1.setUserPhone("1381111****");
        ContactInfo contactInfo2 = new ContactInfo();
        contactInfo2.setUserName("userB");
        contactInfo2.setUserPhone("1382222****");
        contactInfosList.add(contactInfo1);
        contactInfosList.add(contactInfo2);
        request.setContactInfos(contactInfosList);
        // attrs
        //request.setQueueSize(123);
        request.setTaskMaxAttempt(1);
        request.setTaskAttemptInterval(100);
        CreateJobResponse response = client.getAcsResponse(request);
        if (response.getSuccess()) {
            System.out.println("jobId=" + response.getData().getJobId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

创建HTTP任务

```
package com.alibaba.schedulerx.pop;
import com.alibaba.schedulerx.common.domain.HttpAttribute;
import com.alibaba.schedulerx.common.util.JsonUtil;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.CreateJobResponse;
public class CreateHttpJob {
    public static void main(String[] args) throws Exception {
     // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的AccessKeyId，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的AccessKeySecret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        CreateJobRequest request = new CreateJobRequest();
        request.setNamespace("xxxxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobType("http");
        request.setName("testHttpJob");
        request.setDescription("testHttpJob");
        request.setTimeType(1);
        request.setTimeExpression("20 0/5 * * * ?");
        request.setExecuteMode("standalone");
        HttpAttribute httpAttribute = new HttpAttribute();
        httpAttribute.setUrl("http://192.168.0.0:8080/test");
        httpAttribute.setMethod("GET");
        httpAttribute.setTimeout(10); //单位秒
        httpAttribute.setRespKey("code");
        httpAttribute.setRespValue("200");
        request.setContent(JsonUtil.toJson(httpAttribute));
        //POST参数，格式key1=value1&key2=value2。
        request.setParameters("key1=value1&key2=value2");
        //发送请求。
        CreateJobResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("createApp: "+response.getRequestId());
        } else {
            System.out.println(JsonUtil.toJson(response));
        }
    }
}
```

6.10.9. DeleteJob

调用DeleteJob接口删除指定任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteJob	系统规定参数。取值：DeleteJob。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的**应用管理**页面中获取。
JobId	Long	是	92583	任务ID，在控制台的**任务管理**页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	接口状态码。
Message	String	message	附加信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID。
Success	Boolean	true	是否成功。 • **true**：删除任务成功。 • **false**：删除任务失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteJob
&GroupId=testSchedulerx.defaultGroup
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DeleteJobResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <Code>200</Code>
  <Success>true</Success>
</DeleteJobResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
    "Code": 200,
    "Success": true
}
```

示例Demo

```
package com.alibaba.schedulerx.pop;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.DeleteJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.DeleteJobResponse;
public class DeleteJob {
    public static void main(String[] args) throws Exception {
        // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        DeleteJobRequest request = new DeleteJobRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobId((long) 2030221);
        DeleteJobResponse response = client.getAcsResponse(request);
        if (response.getSuccess()) {
            System.out.println("Success: "+response.getSuccess());
        } else {
            System.out.println("Message: "+response.getMessage());
        }
    }
}
```

6.10.10. DeleteWorkflow

调用DeleteWorkflow删除指定工作流。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
WorkflowId	Long	是	111	工作流ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	Your request is denied as lack of ssl protect.	错误消息，仅错误时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	删除工作流是否成功。取值如下： • **true**：删除工作流成功。 • **false**：删除工作流失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteWorkflow
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&WorkflowId=111
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DeleteWorkflowResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>Your request is denied as lack of ssl protect.</Message>
  <Code>200</Code>
  <Success>true</Success>
</DeleteWorkflowResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "Your request is denied as lack of ssl protect.",
    "Code": 200,
    "Success": true
}
```

6.10.11. DisableJob

调用DisableJob停用指定任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	jobid: 92583 not match groupId: testSchedulerx.defaultGroup	错误消息，仅出错时返回错误信息。
RequestId	String	C8E5FB4A-6D8D-424D-9AAA-4FE06BB74FF9	请求唯一ID

名称	类型	示例值	描述
Success	Boolean	true	禁用任务是否成功。取值如下： • **true**：禁用任务成功。 • **false**：禁用任务失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DisableJob
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DisableJobResponse>
  <RequestId>C8E5FB4A-6D8D-424D-9AAA-4FE06BB74FF9</RequestId>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <Code>200</Code>
  <Success>true</Success>
</DisableJobResponse>
```

`JSON` 格式

```
{
    "RequestId": "C8E5FB4A-6D8D-424D-9AAA-4FE06BB74FF9",
    "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
    "Code": 200,
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.DisableJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.DisableJobResponse;
public class DisableJob {
    public static void main(String[] args) throws Exception {
     // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的AccessKeyId，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的AccessKeySecret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        DisableJobRequest request = new DisableJobRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobId(123L);
        DisableJobResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("DisableJob: "+response.getRequestId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

6.10.12. DisableWorkflow

调用DisableWorkflow禁用指定工作流。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
WorkflowId	Long	是	111	工作流ID。

名称	类型	是否必选	示例值	描述
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	Your request is denied as lack of ssl protect.	错误消息，仅错误时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	禁用工作流是否成功。取值如下： • **true**：禁用工作流成功。 • **false**：禁用工作流失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DisableWorkflow
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&WorkflowId=111
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DisableWorkflowResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>Your request is denied as lack of ssl protect.</Message>
  <Code>200</Code>
  <Success>true</Success>
</DisableWorkflowResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "Your request is denied as lack of ssl protect.",
    "Code": 200,
    "Success": true
}
```

6.10.13. EnableJob

调用EnableJob接口启用指定任务。

任务创建完成以后默认启用，所以该功能是在停用任务后使用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	EnableJob	系统规定参数。取值：EnableJob。
JobId	Long	是	92555	任务ID，在控制台的任务管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	jobid: 92583 not match groupId: testSchedulerx.defaultGroup	附加信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBC***	请求唯一ID
Success	Boolean	true	调用任务是否成功。取值如下： • **true**：调用任务成功。 • **false**：调用任务失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=EnableJob
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<EnableJobResponse>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBC***</RequestId>
  <Code>200</Code>
  <Success>true</Success>
</EnableJobResponse>
```

`JSON` 格式

```
{
    "EnableJobResponse": {
        "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
        "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBC***",
        "Code": 200,
        "Success": true
    }
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.EnableJobRequest;
import com.aliyuncs.schedulerx2.model.v20190430.EnableJobResponse;
public class EnableJob {
    public static void main(String[] args) throws Exception {
     // OpenAPI的接入点，具体查看上表支持地域列表以及购买机器地域填写。
        String regionId = "cn-shanghai";
        //鉴权使用的 AccessKeyId，由阿里云官网控制台获取。
        String accessKeyId = "<yourAccessKeyId>";
        //鉴权使用的 AccessKeySecret，由阿里云官网控制台获取。
        String accessKeySecret = "<yourAccessKeySecret>";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.cn-shanghai.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        EnableJobRequest request = new EnableJobRequest();
        request.setNamespace("xxxxxx");
        request.setGroupId("xxxxxx");
        request.setJobId(123L);
        EnableJobResponse response = client.getAcsResponse(request);
        if (!response.getSuccess()) {
            System.out.println(response.getMessage());
            System.out.println("EnableJob: "+response.getRequestId());
        } else {
            System.out.println(response.getMessage());
        }
    }
}
```

6.10.14. EnableWorkflow

调用EnableWorkflow启用指定工作流。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
WorkflowId	Long	是	111	工作流ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	否	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	Your request is denied as lack of ssl protect.	错误信息，仅错误时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	启用工作流是否成功。取值如下： • **true**：启用工作流成功。 • **false**：启用工作流失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=EnableWorkflow
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&WorkflowId=111
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<EnableWorkflowResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>Your request is denied as lack of ssl protect.</Message>
  <Code>200</Code>
  <Success>true</Success>
</EnableWorkflowResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "Your request is denied as lack of ssl protect.",
    "Code": 200,
    "Success": true
}
```

6.10.15. ExecuteJob

调用ExecuteJob接口触发一次任务。所有时间类型的任务， 都可以通过ExecuteJob接口来触发任务。

> **说明** 因为 `JobID` 联合 `ScheduleTime` 是唯一索引，所以同一个任务连续调用ExecuteJob接口的时，每次要sleep一秒，否则任务有可能会失败。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ExecuteJob	系统规定参数。取值：`ExecuteJob`。
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
InstanceParameters	String	是	test	本次触发携带参数，可以是任意字符串，`processor` 代码通过 `context.getInstanceParameters()` 获取，区别于创建任务自定义参数。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。
Message	String	groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	错误消息，仅出错时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108****	请求唯一ID。
Success	Boolean	true	触发任务是否成功。取值如下： • `true`：触发任务成功。 • `false`：触发任务失败。
Data	Object		如果成功，会返回任务实例ID。
JobInstanceId	Long	11111111	任务实例ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ExecuteJob
&JobId=92583
&InstanceParameters=test
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&NamespaceSource=schedulerx
&GroupId=testSchedulerx.defaultGroup
&RegionId=cn-hangzhou
&CheckJobStatus=true
&DesignateType=1
&Worker=test
&Label=test
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ExecuteJobResponse>
    <Code>200</Code>
    <Message>groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9-bbaa-20e90ffc****</Message>
    <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108****</RequestId>
    <Success>true</Success>
    <Data>
        <JobInstanceId>11111111</JobInstanceId>
    </Data>
</ExecuteJobResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Code" : 200,
  "Message" : "groupid not exist groupId: testSchedulerx.defaultGroup namespace: adcfc35d-e2fe-4fe9-bbaa-20e90ffc****",
  "RequestId" : "4F68ABED-AC31-4412-9297-D9A8F0401108****",
  "Success" : true,
  "Data" : {
    "JobInstanceId" : 11111111
  }
}
```

6.10.16. ExecuteWorkflow

调用ExecuteWorkflow触发一次工作流。API类型工作流触发需要调用方法，非API类型也可以调用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
InstanceParameters	String	是	test	工作流实例动态参数，不超过1000字节。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
WorkflowId	Long	是	111	工作流ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
RegionId	String	否	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Data	Struct		如果调用成功，会返回流程实例ID。
WfInstanceId	Long	111111	工作流实例ID
Message	String	Cannot find product according to your domain.	错误消息，仅错误时返回错误信息。

名称	类型	示例值	描述
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	是否成功

示例

请求示例

```
http(s)://[Endpoint]/?Action=ExecuteWorkflow
&GroupId=testSchedulerx.defaultGroup
&InstanceParameters=test
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&WorkflowId=111
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<ExecuteWorkflowResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>Cannot find product according to your domain.</Message>
  <Data>
        <WfInstanceId>111111</WfInstanceId>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</ExecuteWorkflowResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "Cannot find product according to your domain.",
    "Data": {
        "WfInstanceId": 111111
    },
    "Code": 200,
    "Success": true
}
```

6.10.17. GetJobInfo

调用GetJobInfo获取指定Jobid任务详情，通常用来更新任务。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Data	Struct		指定任务信息
JobConfigInfo	Struct		任务配置信息
AttemptInterval	Integer	30	错误重试间隔时间，单位s，默认值为30。
ClassName	String	com.alibaba.test.helloword	任务接口类完整路径，仅是Java类型任务时有该字段。
Content	String	echo "clear" > /home/admin/edas-container/logs/catalina.out	脚本类型任务，具体执行代码。
Description	String	test	任务描述
ExecuteMode	String	standalone	任务执行模式，目前包含以下几种任务执行模式： • **单机运行**：standalone • **广播运行**：broadcatst • **并行计算**：parallel • **内存网格**：grid • **网格计算**：batch • **分片运行**：shard
JarUrl	String	https://test.oss-cn-hangzhou.aliyuncs.com/schedulerX/test.jar	上传到OSS的完整路径。 如果选择JAR包运行，可以将相应JAR包上传到OSS的该路径下。
JobMonitorInfo	Struct		任务监控信息
ContactInfo	Array of ContactInfo		联系人信息
UserName	String	userA	用户名称

名称	类型	示例值	描述
UserPhone	String	1381111****	用户手机号
MonitorConfig	Struct		报警开关以及阈值配置
FailEnable	Boolean	true	失败报警开关。取值如下： • **true**：开启失败报警开关。 • **false**：关闭失败报警开关。
SendChannel	String	sms	报警发送形式，目前只支持sms。
Timeout	Long	12300	超时阈值，单位s，默认7200。
TimeoutEnable	Boolean	true	超时报警开关。取值如下： • **true**：开启超时报警开关。 • **false**：关闭超时报警开关。
TimeoutKillEnable	Boolean	true	超时终止本次触发开关，默认关闭。
MapTaskXAttrs	Struct		高级配置，仅限于并行计算、内存网格和网格计算使用。
ConsumerSize	Integer	5	单机单次触发执行线程数，默认值为5。
DispatcherSize	Integer	5	子任务分发线程数，默认值为5。
PageSize	Integer	100	并行任务单次拉取子任务数，默认值为100。
QueueSize	Integer	10000	子任务队列缓存上限，默认值为10000。
TaskAttemptInterval	Integer	0	子任务失败重试间隔。
TaskMaxAttempt	Integer	0	子任务失败重试次数。
MaxAttempt	Integer	0	错误最大重试次数，根据业务需求填写，默认值为0。
MaxConcurrency	String	1	最大同时运行实例数量，默认值为1， 即上次触发没有运行结束，即使到了运行时刻也不会进行下次触发。
Name	String	helloworld	任务名
Parameters	String	test	用户自定义参数，运行时可以获取。
Status	Integer	1	任务状态。取值如下： • **1**：启用，可以被正常触发。 • **0**：禁用，不会被触发。
TimeConfig	Struct		时间配置信息
Calendar	String	工作日	cron类型可以选择填写自定义日历。
DataOffset	Integer	0	cron类型可以选择时间偏移，单位s。

名称	类型	示例值	描述
TimeExpression	String	0 0/10 * * * ?	时间表达式，目前支持以下几种时间表达类型： • **api**：无时间表达式。 • **fix_rate**：具体固定频率值，如30表示每隔30s触发一次。 • **cron**：标准的cron表达式。 • **second_delay**：固定延迟多少秒执行一次（1s~60s可选）。
TimeType	Integer	1	时间配置类型，目前支持以下几种时间类型： • **cron**：1 • **fix_rate**：3 • **second_delay**：4 • **api**：100
Message	String	jobid: 92583 not match groupId: testSchedulerx.defaultGroup	错误信息，仅错误时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	获取任务详情是否成功。取值如下： • **true**：获取任务详情成功。 • **false**：获取任务详情失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetJobInfo
&GroupId=testSchedulerx.defaultGroup
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<GetJobInfoResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <Data>
        <JobConfigInfo>
              <Status>1</Status>
              <MaxAttempt>0</MaxAttempt>
              <Description>test</Description>
              <Parameters>test</Parameters>
              <JarUrl>https://test.oss-cn-hangzhou.aliyuncs.com/schedulerX/test.jar</JarUrl>
              <MaxConcurrency>1</MaxConcurrency>
              <TimeConfig>
                    <TaskAttemptInterval>0</TaskAttemptInterval>
                    <QueueSize>10000</QueueSize>
```

```
                <PageSize>100</PageSize>
                <DispatcherSize>5</DispatcherSize>
                <TaskMaxAttempt>0</TaskMaxAttempt>
                <Calendar>工作日</Calendar>
                <TimeExpression>0 0/10 * * * ?</TimeExpression>
                <ContactInfo>
                      <UserName>userA</UserName>
                      <UserPhone>1381111****</UserPhone>
                </ContactInfo>
                <ConsumerSize>5</ConsumerSize>
                <DataOffset>0</DataOffset>
                <TimeType>1</TimeType>
                <MonitorConfig>
                      <TimeoutEnable>true</TimeoutEnable>
                      <Timeout>12300</Timeout>
                      <FailEnable>true</FailEnable>
                      <SendChannel>sms</SendChannel>
                      <TimeoutKillEnable>true</TimeoutKillEnable>
                </MonitorConfig>
          </TimeConfig>
          <Name>helloworld</Name>
          <MapTaskXAttrs>
                <TaskAttemptInterval>0</TaskAttemptInterval>
                <QueueSize>10000</QueueSize>
                <PageSize>100</PageSize>
                <DispatcherSize>5</DispatcherSize>
                <TaskMaxAttempt>0</TaskMaxAttempt>
                <Calendar>工作日</Calendar>
                <TimeExpression>0 0/10 * * * ?</TimeExpression>
                <ContactInfo>
                      <UserName>userA</UserName>
                      <UserPhone>1381111****</UserPhone>
                </ContactInfo>
                <ConsumerSize>5</ConsumerSize>
                <DataOffset>0</DataOffset>
                <TimeType>1</TimeType>
                <MonitorConfig>
                      <TimeoutEnable>true</TimeoutEnable>
                      <Timeout>12300</Timeout>
                      <FailEnable>true</FailEnable>
                      <SendChannel>sms</SendChannel>
                      <TimeoutKillEnable>true</TimeoutKillEnable>
                </MonitorConfig>
          </MapTaskXAttrs>
          <JobMonitorInfo>
                <TaskAttemptInterval>0</TaskAttemptInterval>
                <QueueSize>10000</QueueSize>
                <PageSize>100</PageSize>
                <DispatcherSize>5</DispatcherSize>
                <TaskMaxAttempt>0</TaskMaxAttempt>
                <Calendar>工作日</Calendar>
                <TimeExpression>0 0/10 * * * ?</TimeExpression>
                <ContactInfo>
                      <UserName>userA</UserName>
                      <UserPhone>1381111****</UserPhone>
                </ContactInfo>
                <ConsumerSize>5</ConsumerSize>
                <DataOffset>0</DataOffset>
                <TimeType>1</TimeType>
```

```
                    <MonitorConfig>
                          <TimeoutEnable>true</TimeoutEnable>
                          <Timeout>12300</Timeout>
                          <FailEnable>true</FailEnable>
                          <SendChannel>sms</SendChannel>
                          <TimeoutKillEnable>true</TimeoutKillEnable>
                    </MonitorConfig>
              </JobMonitorInfo>
              <Content>echo "clear" &amp;gt; /home/admin/edas-container/logs/catalina.out</Content>
              <ClassName>com.alibaba.test.helloword</ClassName>
              <AttemptInterval>30</AttemptInterval>
              <ExecuteMode>standalone</ExecuteMode>
        </JobConfigInfo>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</GetJobInfoResponse>
```

JSON 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
    "Data": {
        "JobConfigInfo": {
            "Status": 1,
            "MaxAttempt": 0,
            "Description": "test",
            "Parameters": "test",
            "JarUrl": "https://test.oss-cn-hangzhou.aliyuncs.com/schedulerX/test.jar",
            "MaxConcurrency": 1,
            "TimeConfig": {
                "TaskAttemptInterval": 0,
                "QueueSize": 10000,
                "PageSize": 100,
                "DispatcherSize": 5,
                "TaskMaxAttempt": 0,
                "Calendar": "工作日",
                "TimeExpression": "0 0/10 * * * ?",
                "ContactInfo": {
                    "UserName": "userA",
                    "UserPhone": "1381111****"
                },
                "ConsumerSize": 5,
                "DataOffset": 0,
                "TimeType": 1,
                "MonitorConfig": {
                    "TimeoutEnable": true,
                    "Timeout": 12300,
                    "FailEnable": true,
                    "SendChannel": "sms",
                    "TimeoutKillEnable": true
                }
            },
            "Name": "helloworld",
            "MapTaskXAttrs": {
                "TaskAttemptInterval": 0,
                "QueueSize": 10000,
```

```
                "PageSize": 100,
                "DispatcherSize": 5,
                "TaskMaxAttempt": 0,
                "Calendar": "工作日",
                "TimeExpression": "0 0/10 * * * ?",
                "ContactInfo": {
                    "UserName": "userA",
                    "UserPhone": "1381111****"
                },
                "ConsumerSize": 5,
                "DataOffset": 0,
                "TimeType": 1,
                "MonitorConfig": {
                    "TimeoutEnable": true,
                    "Timeout": 12300,
                    "FailEnable": true,
                    "SendChannel": "sms",
                    "TimeoutKillEnable": true
                }
            },
            "JobMonitorInfo": {
                "TaskAttemptInterval": 0,
                "QueueSize": 10000,
                "PageSize": 100,
                "DispatcherSize": 5,
                "TaskMaxAttempt": 0,
                "Calendar": "工作日",
                "TimeExpression": "0 0/10 * * * ?",
                "ContactInfo": {
                    "UserName": "userA",
                    "UserPhone": "1381111****"
                },
                "ConsumerSize": 5,
                "DataOffset": 0,
                "TimeType": 1,
                "MonitorConfig": {
                    "TimeoutEnable": true,
                    "Timeout": 12300,
                    "FailEnable": true,
                    "SendChannel": "sms",
                    "TimeoutKillEnable": true
                }
            },
            "Content": "echo \"clear\" &amp;gt; /home/admin/edas-container/logs/catalina.out",
            "ClassName": "com.alibaba.test.helloword",
            "AttemptInterval": 30,
            "ExecuteMode": "standalone"
        }
    },
    "Code": 200,
    "Success": true
}
```


6.10.18. GetJobInstance

调用GetJobInstance获取指定任务实例详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
JobId	Long	是	92583	任务ID。
JobInstanceId	Long	是	11111111	实例ID。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
RegionId	String	否	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Data	Struct		instance信息
JobInstanceDetail	Struct		任务实例详细信息
DataTime	String	2020-07-27 11:52:10	数据时间
EndTime	String	2020-07-27 11:52:10	任务执行结束时间
Executor	String	A	执行人
InstanceId	Long	11111111	实例ID
JobId	Long	92583	任务ID
Progress	String	complete	实例进度详情
Result	String	success	实例执行结果
ScheduleTime	String	2020-07-27 11:52:10	任务调度时间
StartTime	String	2020-07-27 11:52:10	任务执行开始时间

名称	类型	示例值	描述
Status	Integer	4	实例状态。包含以下几种状态： • 1：等待 • 3：运行中 • 4：成功 • 5：失败 • 9：拒绝 对应枚举类：com.alibaba.schedulerx.common.domain.InstanceStatus
TimeType	Integer	1	实例调度时间类型，包含以下几种时间类型： • cron：1 • fix_rate：3 • second_delay：4 • api：100 对应枚举类：com.alibaba.schedulerx.common.domain.TimeType
TriggerType	Integer	3	触发类型，包含以下几种触发类型： • 1：定时调度正常触发 • 2：数据重刷 • 3：API触发 • 4：用户手动点击重跑 • 5：系统重试（系统异常，如DB异常） 对应枚举类：com.alibaba.schedulerx.common.domain.TriggerType
WorkAddr	String	192.168.0.0:16	被触发客户端IP:Port
Message	String	jobid: 92583 not match groupId: testSchedulerx.defaultGroup	错误消息，仅出错时返回错信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	获取任务实例详情是否成功。取值如下： • true：获取任务实例详情成功。 • false：获取任务实例详情失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetJobInstance
&GroupId=testSchedulerx.defaultGroup
&JobId=92583
&JobInstanceId=11111111
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<GetJobInstanceResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <Data>
        <JobInstanceDetail>
              <Status>4</Status>
              <TriggerType>3</TriggerType>
              <Progress>complete</Progress>
              <EndTime>2020-07-27 11:52:10</EndTime>
              <InstanceId>11111111</InstanceId>
              <WorkAddr>192.168.0.0:16</WorkAddr>
              <StartTime>2020-07-27 11:52:10</StartTime>
              <DataTime>2020-07-27 11:52:10</DataTime>
              <Result>success</Result>
              <TimeType>1</TimeType>
              <Executor>A</Executor>
              <ScheduleTime>2020-07-27 11:52:10</ScheduleTime>
              <JobId>92583</JobId>
        </JobInstanceDetail>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</GetJobInstanceResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
    "Data": {
        "JobInstanceDetail": {
            "Status": 4,
            "TriggerType": 3,
            "Progress": "complete",
            "EndTime": "2020-07-27 11:52:10",
            "InstanceId": 11111111,
            "WorkAddr": "192.168.0.0:16",
            "StartTime": "2020-07-27 11:52:10",
            "DataTime": "2020-07-27 11:52:10",
            "Result": "success",
            "TimeType": 1,
            "Executor": "A",
            "ScheduleTime": "2020-07-27 11:52:10",
            "JobId": 92583
        }
    },
    "Code": 200,
    "Success": true
}
```


6.10.19. GetJobInstanceList

调用GetJobInstanceList获取指定任务ID的执行实例列表。该接口只返回最近10条运行实例列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetJobInstanceList	系统规定参数。取值：GetJobInstanceList。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码

名称	类型	示例值	描述
Data	Struct		instance列表
JobInstanceDetails	Array of JobInstanceDetails		任务实例详细信息
DataTime	String	2020-07-27 11:52:10	数据时间
EndTime	String	2020-07-27 11:52:10	任务执行结束时间
Executor	String	A	执行人
InstanceId	Long	11111111	实例ID
JobId	Long	92583	任务ID
Progress	String	complete	实例进度详情
Result	String	success	实例执行结果
ScheduleTime	String	2020-07-27 11:52:10	任务调度时间
StartTime	String	2020-07-27 11:52:10	任务执行开始时间
Status	Integer	4	实例状态。包含以下几种状态： • 1：等待 • 3：运行中 • 4：成功 • 5：失败 • 9：拒绝 对应枚举类：com.alibaba.schedulerx.common.domain.InstanceStatus
TimeType	Integer	1	实例调度时间类型，包含以下几种类型： • cron：1 • fix_rate：3 • second_delay：4 • api：100 对应枚举类：com.alibaba.schedulerx.common.domain.TimeType
TriggerType	Integer	3	触发类型，包含以下几种触发类型： • 1：定时调度正常触发 • 2：数据重刷 • 3：API触发 • 4：用户手动点击重跑 • 5：系统重试（系统异常，如DB异常） 对应枚举类：com.alibaba.schedulerx.common.domain.TriggerType

名称	类型	示例值	描述
WorkAddr	String	192.168.0.0:16	被触发客户端IP:Port
Message	String	jobid: 92583 not match groupId: testSchedulerx.defaultGroup	错误信息，仅出错时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	获取任务的执行实例列表是否成功。取值如下： • **true**：获取任务的执行实例列表成功。 • **false**：获取任务的执行实例列表失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetJobInstanceList
&GroupId=testSchedulerx.defaultGroup
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<GetJobInstanceListResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>jobid: 92583 not match groupId: testSchedulerx.defaultGroup</Message>
  <Data>
        <JobInstanceDetails>
              <Status>4</Status>
              <TriggerType>3</TriggerType>
              <Progress>complete</Progress>
              <EndTime>2020-07-27 11:52:10</EndTime>
              <InstanceId>11111111</InstanceId>
              <WorkAddr>192.168.0.0:16</WorkAddr>
              <StartTime>2020-07-27 11:52:10</StartTime>
              <DataTime>2020-07-27 11:52:10</DataTime>
              <Result>success</Result>
              <TimeType>1</TimeType>
              <Executor>A</Executor>
              <ScheduleTime>2020-07-27 11:52:10</ScheduleTime>
              <JobId>92583</JobId>
        </JobInstanceDetails>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</GetJobInstanceListResponse>
```

`JSON` 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "jobid: 92583 not match groupId: testSchedulerx.defaultGroup",
    "Data": {
        "JobInstanceDetails": {
            "Status": 4,
            "TriggerType": 3,
            "Progress": "complete",
            "EndTime": "2020-07-27 11:52:10",
            "InstanceId": 11111111,
            "WorkAddr": "192.168.0.0:16",
            "StartTime": "2020-07-27 11:52:10",
            "DataTime": "2020-07-27 11:52:10",
            "Result": "success",
            "TimeType": 1,
            "Executor": "A",
            "ScheduleTime": "2020-07-27 11:52:10",
            "JobId": 92583
        }
    },
    "Code": 200,
    "Success": true
}
```

6.10.20. ListJobs

调用ListJobs获取任务列表。

说明 在调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
      <version>1.0.5</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListJobs	系统规定参数。取值：ListJobs。
GroupId	String	是	DocTest.Group	应用ID，在控制台的**应用管理**页面中获取。
Namespace	String	是	1a72ecb1-b4cc-400a-a71b-20cdec9b****	命名空间，在控制台的**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
JobName	String	否	helloword	任务名称。
Status	String	否	1	任务状态。 • **0**：表示禁用 • **1**：表示启用

返回数据

名称	类型	示例值	描述
Code	Integer	200	请求状态码。
Data	Struct		任务列表信息。
Jobs	Array of Job		任务列表及任务详情。
AttemptInterval	Integer	30	错误重试间隔，单位s，默认值为30。
ClassName	String	com.alibaba.schedulerx.test.helloworld	任务接口类完整路径。 当您的任务是Java任务类型时，才有该字段信息。
Content	String	echo 'hello'	python、shell、go任务类型的脚本代码内容。
Description	String	Test	任务描述。

名称	类型	示例值	描述
ExecuteMode	String	standalone	任务执行模式，可能出现的结果如下： • **standalone**：单机运行 • **broadcast**：广播运行 • **parallel**：并行计算 • **grid**：内存网格 • **batch**：网格计算 • **shard**：分片运行
JarUrl	String	https:doc***.oss-cn-hangzhou.aliyuncs.com/sc-****-D-0.0.2-SNAPSHOT.jar	JAR包的OSS完整路径。
JobId	Long	99341	任务ID。
JobMonitorInfo	Struct		任务监控信息。
ContactInfo	Array of ContactInfo		联系人信息。
UserName	String	userA	用户名称。
UserPhone	String	1381111****	用户手机号。
MonitorConfig	Struct		报警开关以及阈值配置。
FailEnable	Boolean	true	失败报警开关。取值如下： • **true**：开启 • **false**：关闭
SendChannel	String	sms	报警发送形式，目前只支持sms。
Timeout	Long	12300	超时阈值，单位s，默认7200。
TimeoutEnable	Boolean	true	超时报警开关。取值如下： • **true**：开启 • **false**：关闭
TimeoutKillEnable	Boolean	false	超时终止本次触发开关，默认关闭。 • **true**：开启 • **false**：关闭
MapTaskXAttrs	Struct		高级配置，仅限于并行计算、内存网格和网格计算使用。
ConsumerSize	Integer	5	单机单次触发执行线程数，默认值为5。
DispatcherSize	Integer	5	子任务分发线程数，默认值为5。
PageSize	Integer	100	并行任务单次拉取子任务数，默认值为100。
QueueSize	Integer	10000	子任务队列缓存上限，默认值为10000。

名称	类型	示例值	描述
TaskAttemptInterval	Integer	0	子任务失败重试间隔。
TaskMaxAttempt	Integer	0	子任务失败重试次数。
MaxAttempt	Integer	0	错误最大重试次数，根据业务需求填写，默认值为0。
MaxConcurrency	String	1	最大同时运行实例数量，默认值为1， 即上次触发没有运行结束，即使到了运行时刻也不会进行下次触发。
Name	String	helloworld	任务名。
Parameters	String	test	用户自定义参数，运行时可以获取。
Status	Integer	1	任务状态。取值如下： • 1：启用，可以被正常触发。 • 0：禁用，不会被触发。
TimeConfig	Struct		时间配置信息
Calendar	String	工作日	cron类型可以选择填写自定义日历。
DataOffset	Integer	0	cron类型可以选择时间偏移，单位s。
TimeExpression	String	0 0/10 * * * ?	时间表达式，目前支持以下几种时间表达类型： • api：无时间表达式。 • fix_rate：具体固定频率值，如30表示每隔30s触发一次。 • cron：标准的cron表达式。 • second_delay：固定延迟多少秒执行一次（1s~60s可选）。
TimeType	Integer	1	时间配置类型，目前支持以下几种时间类型： • 1：cron • 3：fix_rate • 4：second_delay • 100：api
Message	String	namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSource:null	错误信息，仅出错时返回错误信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB58B	请求唯一ID。
Success	Boolean	true	调用接口是否成功。取值如下： • true：成功 • false：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListJobs
&GroupId=DocTest.Group
&Namespace=1a72ecb1-b4cc-400a-a71b-20cdec9b****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<ListJobsResponse>
  <Message>namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSource:n
ull</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB58B</RequestId>
  <Data>
        <Jobs>
              <Status>1</Status>
              <MaxAttempt>0</MaxAttempt>
              <Parameters>test</Parameters>
              <Description>Test</Description>
              <Content>echo 'hello'</Content>
              <JarUrl>https:doc***.oss-cn-hangzhou.aliyuncs.com/sc-****-D-0.0.2-SNAPSHOT.jar</JarUrl
>
              <MaxConcurrency>1</MaxConcurrency>
              <ClassName>com.alibaba.schedulerx.test.helloworld</ClassName>
              <AttemptInterval>30</AttemptInterval>
              <ExecuteMode>standalone</ExecuteMode>
              <JobId>99341</JobId>
              <Name>helloworld</Name>
              <MapTaskXAttrs>
                    <TaskAttemptInterval>0</TaskAttemptInterval>
                    <QueueSize>10000</QueueSize>
                    <DispatcherSize>5</DispatcherSize>
                    <PageSize>100</PageSize>
                    <TaskMaxAttempt>0</TaskMaxAttempt>
                    <ConsumerSize>5</ConsumerSize>
              </MapTaskXAttrs>
              <TimeConfig>
                    <Calendar>工作日</Calendar>
                    <TimeExpression>0 0/10 * * * ?</TimeExpression>
                    <DataOffset>0</DataOffset>
                    <TimeType>1</TimeType>
              </TimeConfig>
              <JobMonitorInfo>
                    <ContactInfo>
                          <UserName>userA</UserName>
                          <UserPhone>1381111****</UserPhone>
                    </ContactInfo>
                    <MonitorConfig>
                          <TimeoutEnable>true</TimeoutEnable>
                          <Timeout>12300</Timeout>
                          <FailEnable>true</FailEnable>
                          <SendChannel>sms</SendChannel>
                          <TimeoutKillEnable>false</TimeoutKillEnable>
                    </MonitorConfig>
              </JobMonitorInfo>
        </Jobs>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</ListJobsResponse>
```

`JSON` 格式

```
{
    "Message": "namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSou
rce:null",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB58B",
    "Data": {
        "Jobs": {
            "Status": 1,
            "MaxAttempt": 0,
            "Parameters": "test",
            "Description": "Test",
            "Content": "echo 'hello'",
            "JarUrl": "https:doc***.oss-cn-hangzhou.aliyuncs.com/sc-****-D-0.0.2-SNAPSHOT.jar",
            "MaxConcurrency": 1,
            "ClassName": "com.alibaba.schedulerx.test.helloworld",
            "AttemptInterval": 30,
            "ExecuteMode": "standalone",
            "JobId": 99341,
            "Name": "helloworld",
            "MapTaskXAttrs": {
                "TaskAttemptInterval": 0,
                "QueueSize": 10000,
                "DispatcherSize": 5,
                "PageSize": 100,
                "TaskMaxAttempt": 0,
                "ConsumerSize": 5
            },
            "TimeConfig": {
                "Calendar": "工作日",
                "TimeExpression": "0 0/10 * * * ?",
                "DataOffset": 0,
                "TimeType": 1
            },
            "JobMonitorInfo": {
                "ContactInfo": {
                    "UserName": "userA",
                    "UserPhone": "1381111****"
                },
                "MonitorConfig": {
                    "TimeoutEnable": true,
                    "Timeout": 12300,
                    "FailEnable": true,
                    "SendChannel": "sms",
                    "TimeoutKillEnable": false
                }
            }
        }
    },
    "Code": 200,
    "Success": true
}
```


Demo

```
package com.alibaba.schedulerx.pop;
import java.util.List;
import com.alibaba.schedulerx.common.util.JsonUtil;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.ListJobsRequest;
import com.aliyuncs.schedulerx2.model.v20190430.ListJobsResponse;
import com.aliyuncs.schedulerx2.model.v20190430.ListJobsResponse.Data.Job;
public class TestListJobs {
    public static void main(String[] args) {
        //OpenAPI的接入点，具体查看支持地域列表以及购买机器地域填写。
        String regionId = "cn-hangzhou";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "XXXXXXXX";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "XXXXXXXX";
        //产品名称
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写
        String domain ="schedulerx.cn.hangzhou.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        ListJobsRequest request = new ListJobsRequest();
        //命名空间。
        request.setNamespace("433d8b23-06e9-408c-aaaa-90d4d1b****");
        //应用ID。
        request.setGroupId("DocTest.Group");
        ListJobsResponse response;
        try {
            response = client.getAcsResponse(request);
            if (!response.getSuccess()) {
                System.out.println(JsonUtil.toJson(response));
                System.out.println(response.getCode());
            } else {
                System.out.println(JsonUtil.toJson(response));
                List<Job> jobs = response.getData().getJobs();
                for (Job job : jobs) {
                    System.out.println("jobId:" + job.getJobId() + ", name:" + job.getName() + ", st
atus=" + job.getStatus());
                }
            }
        } catch (ServerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClientException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

6.10.21. ListGroups

调用ListGroups获取应用列表。

> **说明** 在调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
      <version>1.0.5</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListGroups	系统规定参数。取值：ListGroups。
Namespace	String	是	1a72ecb1-b4cc-400a-a71b-20cdec9b****	命名空间，在控制台的**命名空间**页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	接口状态码。
Data	Struct		应用列表信息。
AppGroups	Array of AppGroup		应用列表和应用详情。
AppKey	String	a3G77O6NZxq/lyo1NC****==	应用Key。
AppName	String	DocTest	应用名称。
Description	String	Test	应用描述。
GroupId	String	DocTest.Group	应用ID。
Message	String	namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSource:null	错误信息，仅出错时返回错误信息。

名称	类型	示例值	描述
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB58A	请求唯一ID。
Success	Boolean	true	调用接口是否成功。取值如下： • **true**：成功 • **false**：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListGroups
&Namespace=1a72ecb1-b4cc-400a-a71b-20cdec9b****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<ListGroupsResponse>
  <Message>namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSource:null</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB58A</RequestId>
  <Data>
        <AppGroups>
              <Description>Test</Description>
              <AppKey>a3G77O6NZxq/lyo1NC****==</AppKey>
              <GroupId>DocTest.Group</GroupId>
              <AppName>DocTest</AppName>
        </AppGroups>
  </Data>
  <Code>200</Code>
  <Success>true</Success>
</ListGroupsResponse>
```

`JSON` 格式

```
{
    "Message": "namespace can not find namespace: 1a72ecb1-b4cc-400a-a71b-20cdec9b****, namespaceSource:null",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB58A",
    "Data": {
        "AppGroups": {
            "Description": "Test",
            "AppKey": "a3G77O6NZxq/lyo1NC****==",
            "GroupId": "DocTest.Group",
            "AppName": "DocTest"
        }
    },
    "Code": 200,
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import java.util.List;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.ListGroupsRequest;
import com.aliyuncs.schedulerx2.model.v20190430.ListGroupsResponse;
import com.aliyuncs.schedulerx2.model.v20190430.ListGroupsResponse.Data.AppGroup;
public class TestListGroups {
    public static void main(String[] args) {
        //OpenAPI的接入点，具体查看支持地域列表以及购买机器地域填写。
        String regionId = "cn-hangzhou";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "XXXXXXXX";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "XXXXXXXX";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写
        String domain ="schedulerx.cn-hangzhou.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        ListGroupsRequest request = new ListGroupsRequest();
        //命名空间ID。
        request.setNamespace("433d8b23-06e9-408c-aaaa-90d4d1b9****");
        ListGroupsResponse response;
        try {
            response = client.getAcsResponse(request);
            if (!response.getSuccess()) {
                System.out.println(response.getMessage());
            } else {
                List<AppGroup> appGroups = response.getData().getAppGroups();
                for (AppGroup appGroup : appGroups) {
                    System.out.println("groupId=" + appGroup.getGroupId() + ", appKey=" + appGroup.g
etAppKey());
                }
            }
        } catch (ServerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClientException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

6.10.22. ListNamespaces

调用ListNamespaces接口获取命名空间列表。

说明 在调用该接口前，需要在POM文件添加以下依赖：

```
<dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-schedulerx2</artifactId>
      <version>1.0.5</version>
</dependency>
```

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListNamespaces	系统规定参数。取值：ListNamespaces。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	接口状态码。
Data	Struct		命名空间列表信息。
Namespaces	Array of Namespace		命名空间列表与详情。
Description	String	test	命名空间描述。
Name	String	doc	命名空间名称。
Uld	String	1a72ecb1-b4cc-400a-a71b-20cdec9b****	命名空间ID。
Message	String	You have an error in your SQL syntax	错误信息，仅出错时返回错误信息。
RequestId	String	71BCC0E3-64B2-4B63-A870-AFB64EBCB58C	请求唯一ID。
Success	Boolean	true	调用接口是否成功。取值如下： • **true**：成功 • **false**：失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListNamespaces
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<ListNamespacesResponse>
  <Message>You have an error in your SQL syntax</Message>
  <RequestId>71BCC0E3-64B2-4B63-A870-AFB64EBCB58C</RequestId>
  <Data>
        <Namespaces>
              <UId>1a72ecb1-b4cc-400a-a71b-20cdec9b****</UId>
              <Description>test</Description>
              <Name>doc</Name>
        </Namespaces>
  </Data>
  <Code></Code>
  <Success>true</Success>
</ListNamespacesResponse>
```

`JSON` 格式

```
{
    "Message": "You have an error in your SQL syntax",
    "RequestId": "71BCC0E3-64B2-4B63-A870-AFB64EBCB58C",
    "Data": {
        "Namespaces": {
            "UId": "1a72ecb1-b4cc-400a-a71b-20cdec9b****",
            "Description": "test",
            "Name": "doc"
        }
    },
    "Code": "",
    "Success": true
}
```

Demo

```
package com.alibaba.schedulerx.pop;
import java.util.List;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.schedulerx2.model.v20190430.ListNamespacesRequest;
import com.aliyuncs.schedulerx2.model.v20190430.ListNamespacesResponse;
import com.aliyuncs.schedulerx2.model.v20190430.ListNamespacesResponse.Data.Namespace;
public class TestListNamespaces {
    public static void main(String[] args) {
        //OpenAPI的接入点，具体查看支持地域列表以及购买机器地域填写。
        String regionId = "cn-test";
        //鉴权使用的AccessKey ID，由阿里云官网控制台获取。
        String accessKeyId = "XXXXXXXX";
        //鉴权使用的AccessKey Secret，由阿里云官网控制台获取。
        String accessKeySecret = "XXXXXXXX";
        //产品名称。
        String productName ="schedulerx2";
        //对照支持地域列表选择Domain填写。
        String domain ="schedulerx.aliyuncs.com";
        //构建OpenAPI客户端。
        DefaultProfile.addEndpoint(regionId, productName, domain);
        DefaultProfile defaultProfile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySe
cret);
        DefaultAcsClient client = new DefaultAcsClient(defaultProfile);
        ListNamespacesRequest request = new ListNamespacesRequest();
        ListNamespacesResponse response;
        try {
            response = client.getAcsResponse(request);
            if (!response.getSuccess()) {
                System.out.println(response.getMessage());
            } else {
                List<Namespace> namespaces = response.getData().getNamespaces();
                for (Namespace namespace : namespaces) {
                    System.out.println("namespace uid=" + namespace.getUId());
                }
            }
        } catch (ServerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClientException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

6.10.23. StopInstance

调用StopInstance终止某次正在运行的实例。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	StopInstance	系统规定参数。取值：StopInstance。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
InstanceId	Long	是	11111111	运行实例ID。
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
RegionId	String	是	cn-hangzhou	地域ID。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码
Message	String	Your request is denied as lack of ssl protect.	错误信息，仅错误时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID
Success	Boolean	true	终止任务运行是否成功。取值如下： • **true**：终止任务运行成功。 • **false**：终止任务运行失败。

示例

请求示例

```
http(s)://[Endpoint]/?Action=StopInstance
&GroupId=testSchedulerx.defaultGroup
&InstanceId=11111111
&JobId=92583
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<StopInstanceResponse>
  <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
  <Message>Your request is denied as lack of ssl protect.</Message>
  <Code>200</Code>
  <Success>true</Success>
</StopInstanceResponse>
```

JSON 格式

```
{
    "RequestId": "4F68ABED-AC31-4412-9297-D9A8F0401108",
    "Message": "Your request is denied as lack of ssl protect.",
    "Code": 200,
    "Success": true
}
```

6.10.24. UpdateJob

调用UpdateJob接口更新任务配置信息。默认先调用获取任务找到对应字段进行修改。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	UpdateJob	系统规定参数。取值：**UpdateJob**。
Name	String	是	helloword	任务名。
Description	String	否	test	任务描述。
ExecuteMode	String	是	standalone	任务执行模式，目前支持以下几种任务执行模式： • **单机运行**：standalone • **广播运行**：broadcatst • **并行计算**：parallel • **内存网格**：grid • **网格计算**：batch • **分片运行**：shard
Content	String	否	echo 'hello'	任务类型为**python**、**shell**、**go**的必填脚本内容。
Parameters	String	否	test	用户自定义参数，运行时可以获取。
MaxConcurrency	Integer	否	1	最大同时运行实例数量，默认值为1，即上次触发没有运行结束，不会进行下次触发即使到了运行时刻。
MaxAttempt	Integer	否	0	错误最大重试次数，根据业务需求填写。
AttemptInterval	Integer	否	30	错误重试间隔时间，单位s，默认值为30。
ClassName	String	否	com.alibaba.test.helloworld	任务接口类完整路径。 是Java任务类型时，才有该字段且必须填写完整路径。
JarUrl	String	否	暂不支持，不用填写	上传到OSS的完整路径。 如果选择JAR包运行，可以将相应JAR包上传到OSS的该路径下。

名称	类型	是否必选	示例值	描述
PageSize	Integer	否	100	并行网格任务高级配置，单次拉取子任务数，默认值为100。
ConsumerSize	Integer	否	5	并行网格任务高级配置，单机单次触发执行线程数，默认值为5。
QueueSize	Integer	否	10000	并行网格任务高级配置，子任务队列缓存上限，默认值为10000。
DispatcherSize	Integer	否	5	并行网格任务高级配置，子任务分发线程数，默认值为5。
TimeType	Integer	是	1	时间配置类型，目前支持以下几种配置类型： • **cron**：1 • **fix_rate**：3 • **second_delay**：4 • **api**：100
TimeExpression	String	否	30	时间表达式，根据选择的时间类型设置时间表达式。 • **cron**：填写标准的cron表达式，支持在线验证。 • **api**：无时间表达式。 • **fixed_rate**：填写具体固定频率值，单位s。如30表示每隔30s触发一次。 • **second_delay**：填写固定延迟多少秒执行一次（1s~60s可选）。
Calendar	String	否	工作日	cron类型可以选择填写自定义日历。
DataOffset	Integer	否	2400	cron类型可以选择时间偏移，单位s。
TimeoutEnable	Boolean	否	true	超时报警开关。取值如下： • **true**：开启超时报警开关。 • **false**：关闭超时报警开关。
Timeout	Long	否	7200	超时阈值，单位s。
TimeoutKillEnable	Boolean	否	true	超时终止本次触发开关。取值如下： • **true**：开启超时终止开关。 • **false**：关闭超时终止开关。
FailEnable	Boolean	否	true	失败报警开关。取值如下： • **true**：开启失败报警开关。 • **false**：关闭失败报警开关。
MissWorkerEnable	Boolean	否	true	是否开启无可用机器告警。 • **true**：开启无可用机器告警开关。 • **false**：关闭无可用机器告警开关。

名称	类型	是否必选	示例值	描述
SendChannel	String	否	sms	报警发送形式，目前只支持sms。
Namespace	String	是	adcfc35d-e2fe-4fe9-bbaa-20e90ffc****	命名空间ID，在控制台的命名空间页面中获取。
GroupId	String	是	testSchedulerx.defaultGroup	应用ID，在控制台的应用管理页面中获取。
NamespaceSource	String	否	schedulerx	特殊第三方才需要填写。
RegionId	String	是	cn-hangzhou	地域ID。
JobId	Long	是	92583	任务ID，在控制台的任务管理页面中获取。
TaskMaxAttempt	Integer	否	0	并行网格任务高级配置，子任务失败重试次数。
TaskAttemptInterval	Integer	否	0	并行网格任务高级配置，子任务失败重试间隔。
ContactInfo.N.UserPhone	String	否	1381111****	用户手机号。
ContactInfo.N.UserName	String	否	userA	用户名。
ContactInfo.N.UserMail	String	否	test***@***.com	用户邮箱。
ContactInfo.N.Ding	String	否	67***798	用户钉钉号。

说明 在调用UpdateJob更新调度任务时，会删除之前对应参数的配置，而非保留。所以，请先调用GetJobInfo，获取目标任务此前的配置，再根据实际情况进行配置。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。
Message	String	job type is java className can not be blank	附加信息，仅出错时返回错误信息。
RequestId	String	4F68ABED-AC31-4412-9297-D9A8F0401108	请求唯一ID。
Success	Boolean	true	是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=UpdateJob
&Name=helloword
&Description=test
&ExecuteMode=standalone
&Content=echo 'hello'
&Parameters=test
&MaxConcurrency=1
&MaxAttempt=0
&AttemptInterval=30
&ClassName=com.alibaba.test.helloworld
&JarUrl=暂不支持，不用填写
&PageSize=100
&ConsumerSize=5
&QueueSize=10000
&DispatcherSize=5
&TimeType=1
&TimeExpression=30
&Calendar=工作日
&DataOffset=2400
&TimeoutEnable=true
&Timeout=7200
&TimeoutKillEnable=true
&FailEnable=true
&MissWorkerEnable=true
&SendChannel=sms
&Namespace=adcfc35d-e2fe-4fe9-bbaa-20e90ffc****
&GroupId=testSchedulerx.defaultGroup
&NamespaceSource=schedulerx
&RegionId=cn-hangzhou
&JobId=92583
&TaskMaxAttempt=0
&TaskAttemptInterval=0
&ContactInfo=[{"UserPhone":"1381111****","UserName":"userA","UserMail":"test***@***.com","Ding":"67*
**798"}]
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<UpdateJobResponse>
    <Code>200</Code>
    <Message>job type is java className can not be blank</Message>
    <RequestId>4F68ABED-AC31-4412-9297-D9A8F0401108</RequestId>
    <Success>true</Success>
</UpdateJobResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Code" : 200,
  "Message" : "job type is java className can not be blank",
  "RequestId" : "4F68ABED-AC31-4412-9297-D9A8F0401108",
  "Success" : true
}
```

6.11. 配置参考

6.11.1. JobContext参数说明

本文介绍JobContext的参数及说明。

参数	解释
long jobId	任务ID
long jobInstanceId	任务实例ID
long wfInstanceId	工作流实例ID
long taskId	分布式任务子任务ID，根任务是0。
DateTime scheduleTime	实例的计划调度时间
DateTime dataTime	实例的数据时间
String jobType	任务类型
String taskName	子任务名称
Object task	子任务body
String jobParameters	控制台配置的静态任务参数。jobParameters有大小限制，不能超过10000字节。
String instanceParameters	通过API触发的动态实例参数
int maxAttempt	实例最大重试次数
int attempt	实例当前重试次数
int taskMaxAttempt	子任务最大重试次数
int taskAttempt	子任务当前重试次数
`List<JobInstanceData> upstreamData`	工作流实例的上游数据，可能有多个上游，`List<JobInstanceData>`格式。
Long shardingId	分片ID，分片模型和广播模型适用。
String shardingParameter	分片参数，分片模型适用。
int shardingNum	分片数量，分片模型和广播模型适用。

6.11.2. 任务管理高级配置参数说明

本文介绍任务管理中的高级配置参数。
任务管理高级配置参数说明如下：

参数	适用的执行模式	解释	默认值
实例失败重试次数	通用	任务运行失败自动重试的次数。	0
实例失败重试间隔	通用	每次失败重试的间隔。单位：秒。	30
实例并发数	通用	同一个Job同一时间运行的实例个数。1表示不允许重复执行。	1
子任务单机并发数	• 并行计算 • 内存网格 • 网格计算	分布式模型，单台机器并发消费子任务的个数。	5
子任务失败重试次数	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的次数。	0
子任务失败重试间隔	• 并行计算 • 内存网格 • 网格计算	分布式模型，子任务失败自动重试的间隔。单位：秒。	0
子任务分发方式	• 并行计算 • 内存网格 • 网格计算	• **推模型**：每台机器平均分配子任务。 • **拉模型**： 每台机器主动拉取子任务，没有木桶效应。拉取过程中，所有子任务会缓存在Master节点，对内存有压力，建议子任务数不超过10,000。	推模型
子任务单次拉取数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点每次向Master节点拉取多少个子任务。	5
子任务队列容量（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	Slave节点缓存子任务的队列大小。	10
子任务全局并发数（仅适用于拉模型）	• 并行计算 • 内存网格 • 网格计算	分布式拉模型支持全局子任务并发数，可以进行限流。	1,000

6.11.3. SchedulerxWorker配置参数说明

本文介绍SchedulerxWorker配置参数。
SchedulerxWorker配置参数说明如下：

参数	解释	默认值
`setEndpoint(String endpoint)`	设置每个Region的地址服务器地址。	无
`setNamespace(String namespace)`	设置命名空间的ID。	无
`setGroupId(String groupId)`	前端先创建应用，客户端接入的时候填写**应用ID**（appKey）。	无
`setAliyunAccessKey(String aliyunAccessKey)`	设置阿里云账号的AccessKey ID，使用appKey后可以不用该配置。	无
`setAliyunSecretKey(String aliyunSecretKey)`	设置阿里云账号的AccessKey Secret，使用appKey后可以不用该配置。	无
`setEnableBatchWork(boolean enableBatchWork)`	是否启用网格计算，默认启用。 • 如果未使用网格计算，不建议设置。 • 如果使用了网格计算，建议开启，否则每台机器的第一次触发会至少延迟20秒才运行。	true
`setHost(String host)`	如果客户端有多个网卡或者VPN环境，默认获取的本机IP可能不对，可以通过该接口自己设置。	无
`setPort(int port)`	用户可以自定义客户端的监听端口	无
`setClassLoader(ClassLoader userClassLoader)`	非Spring应用，使用其他第三方框架，可能需要设置 `custom class loader` 。	无
`setBlockAppStart(boolean block)`	SchedulerX初始化失败，是否block应用进程启动	无
`setEnableUnits(String units)`	单元隔离白名单，多个单元以半角逗号（,）分隔，不允许包含空格。例如worker.setEnableUnites("center,zbyun");	无
`setEnableSites(String sites)`	机房隔离白名单，多个机房以半角逗号（,）分隔，不允许包含空格。例如worker.setEnableSites("center.na61,center.na62");	无
`setDisableUnits(String units)`	单元隔离黑名单，如果设置了白名单，以白名单为准，黑名单失效。多个单元以半角逗号（,）分隔，不允许包含空格。	无

参数	解释	默认值
`setDisableSites(String sites)`	机房隔离黑名单，如果设置了白名单，以白名单为准，黑名单失效。多个机房以半角逗号（,）分隔，不允许包含空格。	无
`setSlsCollectorEnable(boolean enable)`	是否启用SLS日志收集功能。	True
`setShareContainerPool(boolean shareConatinerPool)`	客户端所有任务执行是否共享线程池，推荐大量任务高并发调度的场景开启。	False
`setSharePoolSize(intsharePoolSize)`	如果开启共享线程池，可以自定义线程池大小。	64
`setLabel(String label)`	客户端启动的时候可以设置Label，任务管理指定机器的时候可以指定Label执行。应用于灰度、压测等场景。	无
`setMapMasterStatusCheckInterval(int interval)`	设置Map模型检测所有子任务结束的频率，单位毫秒。如果是秒级别任务，需要加快调度频率，可以设置。	3000
`setEnableSecondDelayCycleIntervalMs(boolean enable)`	设置second_delay延迟的单位为毫秒。如果把这个值设置为true，控制台设置的秒级别延迟将会变成毫秒，可以加快调度频率。	false

6.12. 常见问题

6.12.1. 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，Group ID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EDAS控制台。
   ii. 在左侧导航栏选择**组件中心 > 分布式任务调度（新）**，然后单击**应用管理**。
   iii. 在**应用管理**页面顶部菜单栏选择地域，在页面中选择目标命名空间。

iv. 在**应用管理**页面，单击**创建应用**，根据需要创建应用。

应用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应用ID，如 `xxx.defaultGroup` 。
- 如果使用兼容方案，应用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。

v. 创建完成后，返回**应用管理**页面查看任务分组是否成功。

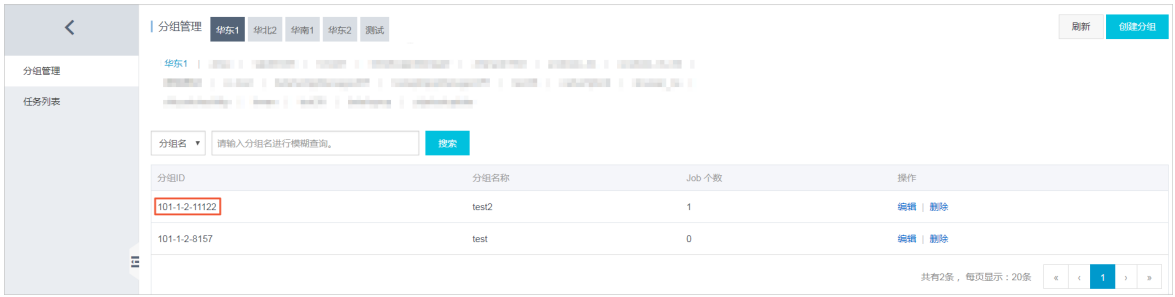
2. 下载迁移工具（JAR包），通过命令行进入迁移工具所在目录，并执行 `java -jar schedulerx-migrate-1.0.20190729.jar` 命令。

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。

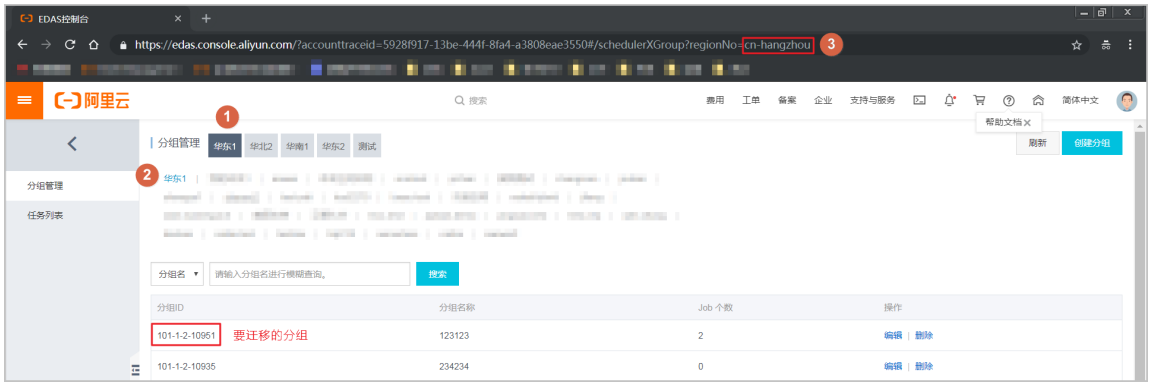
   分组ID可以在Scheduler 1.0 分组管理页面查看。



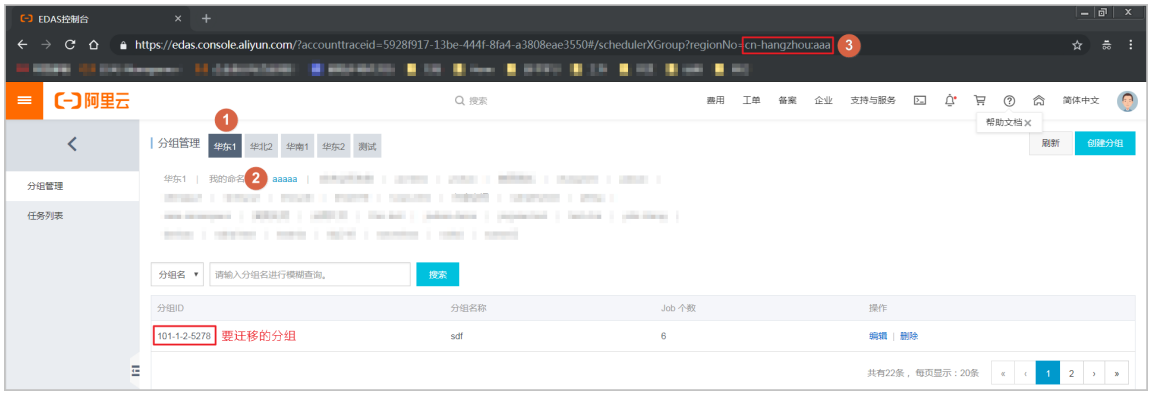
3. 输入分布式任务系统1.0迁移分组所在的 `regionId` （包含对应namespace的key）。

   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` 。

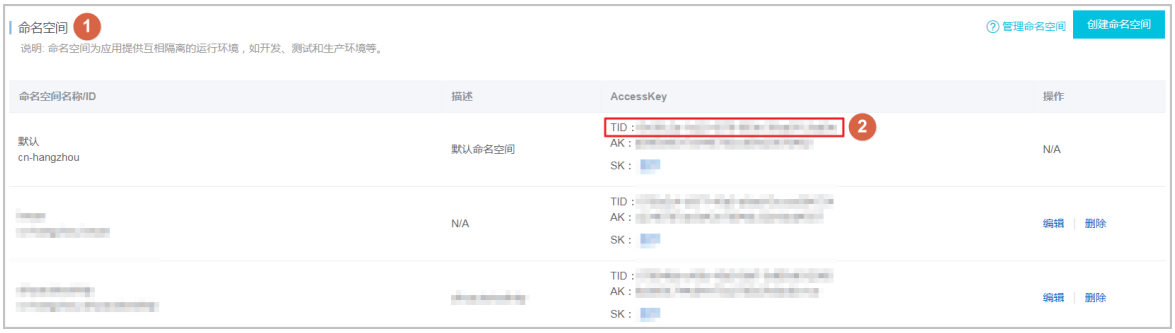


- 如果在非默认命名空间中的任务分组，`regionNo`会包含Region和Namespace，如`cn-hangzhou:aaa`。



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任务分组的云账号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` 。

   填写在使用应用管理设置的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分组的云账号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AccessKey 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.
3.
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` 。

   填写在使用应用管理设置的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对应应用**应用key**。

   输入上一步中应用管理列表里面 `Group_ID` 对应的**应用key**。

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.

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.
2.
3.
4. 输入要迁入到分布式系统2.0的 `regionName` 。
5. 输入迁入到分布式系统2.0分组的 `Group_ID` 。

   填写在使用应用管理设置的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.
7.
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。

6.12.2. 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。

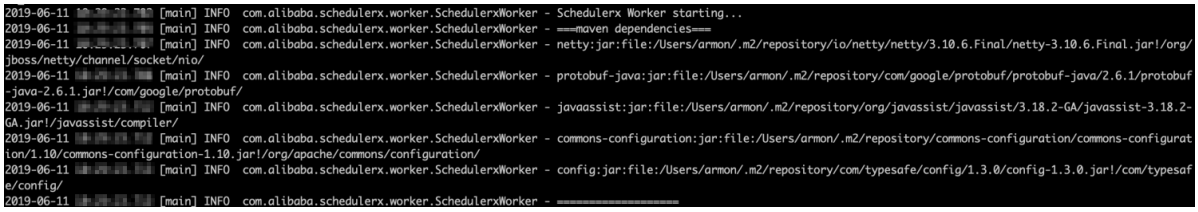
- 如果日志包含 `groupId is not existed` ，说明是配置问题。在*worker.log*里搜索*Schedulerx WorkerConfig*，可以看到当前的配置，然后和控制台对比、确认。

  > **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。

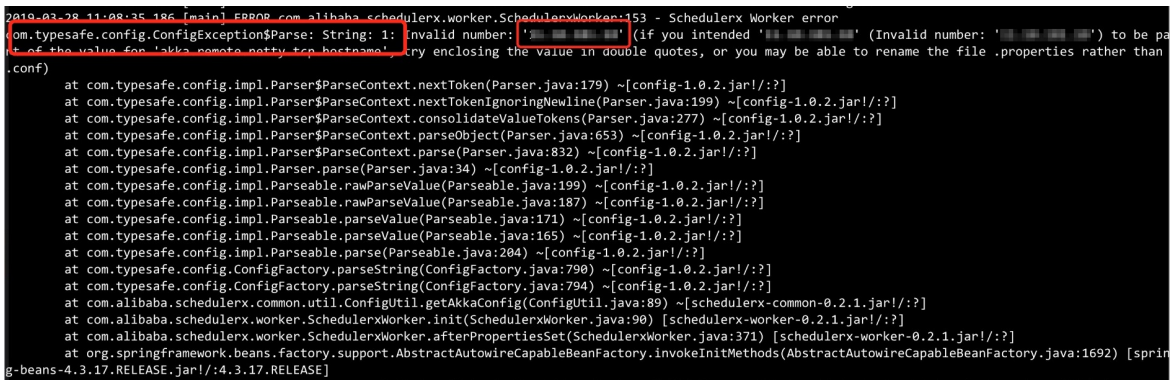
JAR包	版本
guava	20.0
com.typesafe.config	1.3.0
protobuf-java	2.6.1
io.netty	3.10.6.Final
javaassist	3.21.0-GA
hessian	4.0.51
commons-configuration	1.10
commons-validator	1.4.0

- 示例
  下面列出几个典型的JAR冲突现象：

- guava冲突

```
2019-06-06 15:09:33.928 [19478_12371-akka.actor.thread-dispatcher-container-114] ERROR com.alibaba.schedulerx.worker.logcollector.LogCollectorFactory -
java.lang.RuntimeException: java.lang.reflect.InvocationTargetException
        at com.alibaba.schedulerx.common.util.ReflectionUtil.newInstance(ReflectionUtil.java:33) ~[schedulerx-common-1.0.2.jar:?]
        at com.alibaba.schedulerx.common.util.ReflectionUtil.getInstanceByClassName(ReflectionUtil.java:65) ~[schedulerx-common-1.0.2.jar:?]
        at com.alibaba.schedulerx.worker.logcollector.LogCollectorFactory.get(LogCollectorFactory.java:38) [schedulerx-worker-1.0.3.jar:?]
        at com.alibaba.schedulerx.worker.actor.ContainerActor.<init>(ContainerActor.java:56) [schedulerx-worker-1.0.3.jar:?]
        at sun.reflect.GeneratedConstructorAccessor116.newInstance(Unknown Source) [?:1.8.0_201]
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) [?:1.8.0_201]
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423) [?:1.8.0_201]
        at java.lang.Class.newInstance(Class.java:442) [?:1.8.0_201]
        at akka.util.Reflect$.instantiate(Reflect.scala:44) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.NoArgsReflectConstructor.produce(IndirectActorProducer.scala:105) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.Props.newActor(Props.scala:213) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.ActorCell.newActor(ActorCell.scala:562) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.ActorCell.create(ActorCell.scala:588) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.ActorCell.invokeAll$1(ActorCell.scala:461) [akka-actor_2.11-2.4.20.jar:?]
        at akka.actor.ActorCell.systemInvoke(ActorCell.scala:483) [akka-actor_2.11-2.4.20.jar:?]
        at akka.dispatch.Mailbox.processAllSystemMessages(Mailbox.scala:282) [akka-actor_2.11-2.4.20.jar:?]
        at akka.dispatch.Mailbox.run(Mailbox.scala:223) [akka-actor_2.11-2.4.20.jar:?]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [?:1.8.0_201]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [?:1.8.0_201]
        at java.lang.Thread.run(Thread.java:748) [?:1.8.0_201]
Caused by: java.lang.reflect.InvocationTargetException
        at sun.reflect.GeneratedConstructorAccessor113.newInstance(Unknown Source) ~[?:?]
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) ~[?:1.8.0_201]
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423) ~[?:1.8.0_201]
        at com.alibaba.schedulerx.common.util.ReflectionUtil.newInstance(ReflectionUtil.java:31) ~[schedulerx-common-1.0.2.jar:?]
        ... 19 more
Caused by: java.lang.NoSuchMethodError: com.google.common.hash.Hashing.farmHashFingerprint64()Lcom/google/common/hash/HashFunction;
        at com.aliyun.openservices.aliyun.log.producer.internals.Utils.generateProducerHash(Utils.java:31) ~[aliyun-log-producer-0.2.0.jar:?]
        at com.aliyun.openservices.aliyun.log.producer.LogProducer.<init>(LogProducer.java:77) ~[aliyun-log-producer-0.2.0.jar:?]
        at com.alibaba.schedulerx.worker.logcollector.SlsLogCollector.<init>(SlsLogCollector.java:66) ~[schedulerx-worker-1.0.3.jar:?]
        at sun.reflect.GeneratedConstructorAccessor113.newInstance(Unknown Source) ~[?:?]
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45) ~[?:1.8.0_201]
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423) ~[?:1.8.0_201]
        at com.alibaba.schedulerx.common.util.ReflectionUtil.newInstance(ReflectionUtil.java:31) ~[schedulerx-common-1.0.2.jar:?]
        ... 19 more
(END)
```

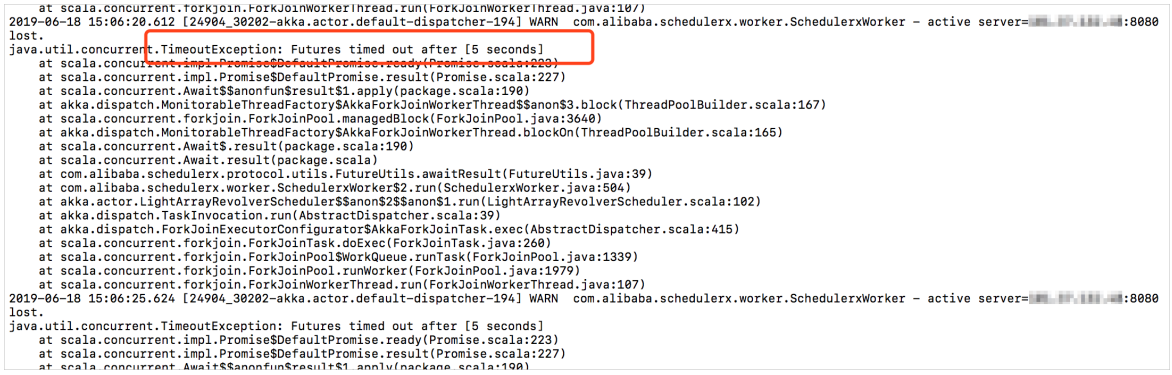
- com.typesafe.config包冲突



```
2019-03-28 11:08:35.186 [main] ERROR com.alibaba.schedulerx.worker.SchedulerxWorker:153 - Schedulerx Worker error
com.typesafe.config.ConfigException$Parse: String: 1: Invalid number: '[redacted]' (if you intended '[redacted]' (Invalid number: '[redacted]') to be pa
t of the value for 'akka.remote.netty.tcp.hostname', try enclosing the value in double quotes, or you may be able to rename the file .properties rather than
.conf)
        at com.typesafe.config.impl.Parser$ParseContext.nextToken(Parser.java:179) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parser$ParseContext.nextTokenIgnoringNewline(Parser.java:199) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parser$ParseContext.consolidateValueTokens(Parser.java:277) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parser$ParseContext.parseObject(Parser.java:653) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parser$ParseContext.parse(Parser.java:832) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parser.parse(Parser.java:34) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parseable.rawParseValue(Parseable.java:199) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parseable.rawParseValue(Parseable.java:187) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parseable.parseValue(Parseable.java:171) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parseable.parseValue(Parseable.java:165) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.impl.Parseable.parse(Parseable.java:204) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.ConfigFactory.parseString(ConfigFactory.java:790) ~[config-1.0.2.jar!/:?]
        at com.typesafe.config.ConfigFactory.parseString(ConfigFactory.java:794) ~[config-1.0.2.jar!/:?]
        at com.alibaba.schedulerx.common.util.ConfigUtil.getAkkaConfig(ConfigUtil.java:89) ~[schedulerx-common-0.2.1.jar!/:?]
        at com.alibaba.schedulerx.worker.SchedulerxWorker.init(SchedulerxWorker.java:90) [schedulerx-worker-0.2.1.jar!/:?]
        at com.alibaba.schedulerx.worker.SchedulerxWorker.afterPropertiesSet(SchedulerxWorker.java:371) [schedulerx-worker-0.2.1.jar!/:?]
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1692) [sprin
g-beans-4.3.17.RELEASE.jar!/:4.3.17.RELEASE]
```

- 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:?]
```

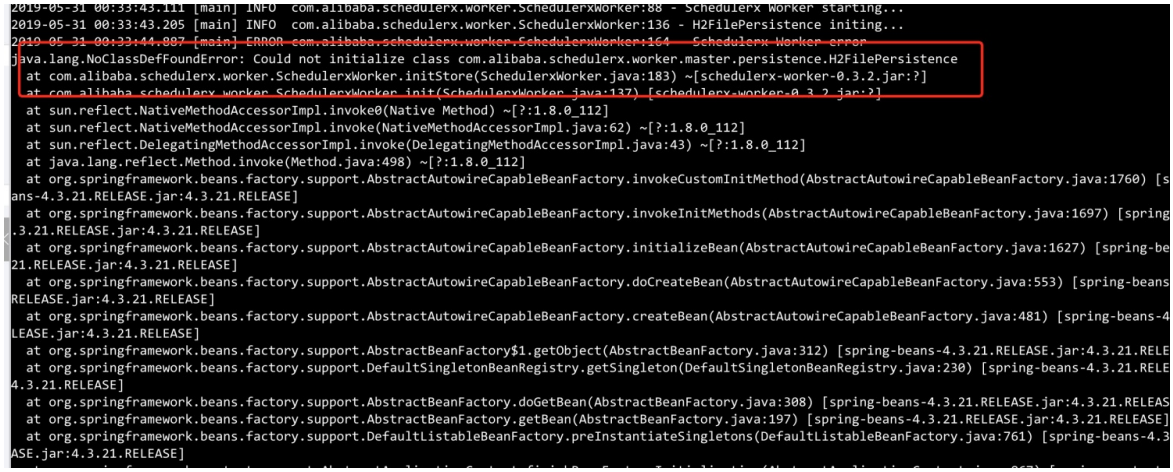
- Hessian冲突

```
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)] timeout,
cost=5004ms
```

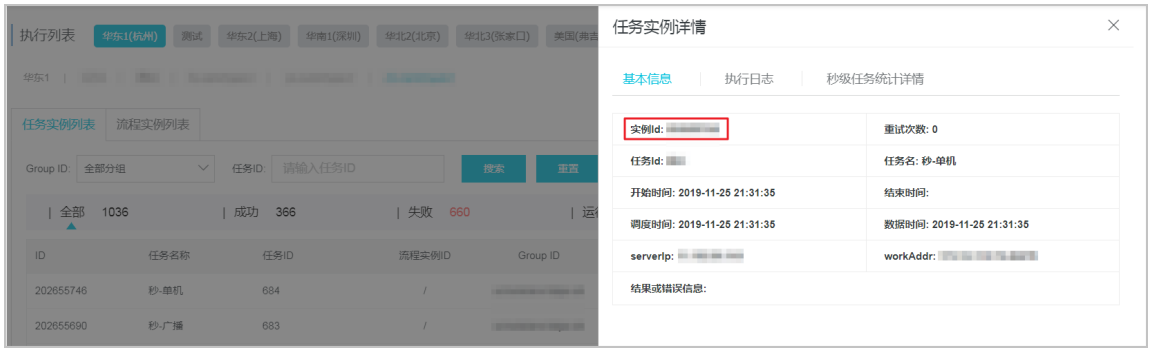
- 可能的原因
  - JAR包冲突
  - 机器负载过高或者重启
- 解决办法
  查看*worker.log*中有没有异常日志。
  - 如果有，可能是JAR包冲突，处理详情请参见JAR包冲突。
  - 如果没有，可能是机器负载过高或者重启造成的，请检查机器。

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。

```
$jstack 29191 |grep 58903617 -A 200
"Schedulerx-Container-Thread-58903617-0" #4093 prio=5 os_prio=0 tid=[redacted] nid=[redacted] waiting on condition [0x00002ad443101000]
   java.lang.Thread.State: WAITING (parking)
  at sun.misc.Unsafe.park(Native Method)
  - parking to wait for  <0x000000072b837ae0> (a java.util.concurrent.CountDownLatch$Sync)
  at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
  at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)
  at java.util.concurrent.locks.AbstractQueuedSynchronizer.doAcquireSharedInterruptibly(AbstractQueuedSynchronizer.java:997)
  at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireSharedInterruptibly(AbstractQueuedSynchronizer.java:1304)
  at java.util.concurrent.CountDownLatch.await(CountDownLatch.java:231)
  at com.aliyun.ticket.importworker.WOImportJob.startImport(WOImportJob.java:353)
  at com.aliyun.ticket.importworker.WOImportJob.doImportForAll(WOImportJob.java:242)
  at com.aliyun.ticket.importworker.WOImportJob.process(WOImportJob.java:163)
  at com.alibaba.schedulerx.worker.container.ThreadContainer.start(ThreadContainer.java:90)
  at com.alibaba.schedulerx.worker.container.ThreadContainer.run(ThreadContainer.java:60)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
  at java.lang.Thread.run(Thread.java:756)

"Container-Batch-Statues-Retrieve-Thread-58903617" #4092 prio=5 os_prio=0 tid=[redacted] nid=[redacted] waiting on condition [0x00002ad44fe0e000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(Native Method)
  at com.alibaba.schedulerx.worker.batch.BaseReqHandler$2.run(BaseReqHandler.java:74)
  at java.lang.Thread.run(Thread.java:756)

"TDDL-Druid-ConnectionPool-DestroyScheduler--2-thread-237" #4052 daemon prio=5 os_prio=0 tid=[redacted] nid=[redacted] waiting on condition [0x00002ad462e00
   java.lang.Thread.State: TIMED_WAITING (parking)
  at sun.misc.Unsafe.park(Native Method)
  - parking to wait for  <0x00000007436c7190> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
  at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)
  at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2078)
  at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.poll(ScheduledThreadPoolExecutor.java:1129)
  at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.poll(ScheduledThreadPoolExecutor.java:809)
  at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1066)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
```

- 否则，请联系SchedulerX技术支持人员。

报slf4j.Slf4jMDC异常

- 现象

```
Exception in thread "TDDL-BufferedStatLogWriter-Flush-Executor" java.lang.NoClassDefFoundError: Could not initialize class org.slf4j.MDC
  at com.taobao.tddl.common.utils.logger.slf4j.Slf4jMDC.remove(Slf4jMDC.java:26)
  at com.taobao.tddl.common.utils.logger.MDC.remove(MDC.java:47)
  at com.taobao.tddl.monitor.stat.BufferedLogWriter$1.run(BufferedLogWriter.java:151)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:627)
  at java.lang.Thread.run(Thread.java:861)
Exception in thread "TDDL-BufferedStatLogWriter-Flush-Executor" java.lang.NoClassDefFoundError: Could not initialize class org.slf4j.MDC
  at com.taobao.tddl.common.utils.logger.slf4j.Slf4jMDC.remove(Slf4jMDC.java:26)
  at com.taobao.tddl.common.utils.logger.MDC.remove(MDC.java:47)
  at com.taobao.tddl.monitor.stat.BufferedLogWriter$1.run(BufferedLogWriter.java:151)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:627)
  at java.lang.Thread.run(Thread.java:861)
```

- 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"
        }
    ]
}
```

6.12.3. 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清除。

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。

7.应用平台管理

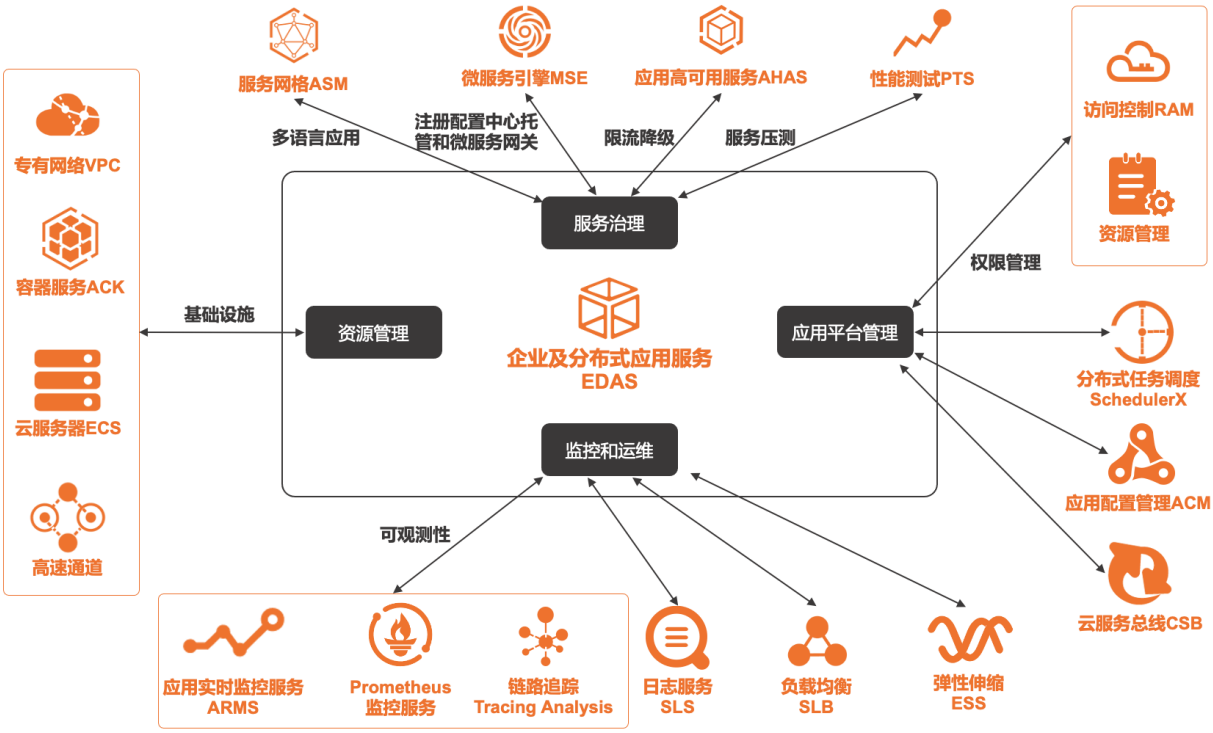
7.1. 云服务集成

7.1.1. 云服务集成概述

EDAS作为应用的一站式PaaS平台，集成了众多阿里云服务，以产品和EDAS内部组件两种形态为EDAS中的应用提供各个层面、维度的功能。

云服务概览

EDAS通过集成其它云服务，提供了基础设施管理、服务治理、监控、运维和权限管理等功能。



EDAS集成所需的云服务包含两种形式：产品集成和组件集成。

- 产品集成：以独立产品的形式集成，相关资源、功能由各产品（非EDAS）管理。
- 组件集成：以EDAS的服务组件形式集成，为EDAS提供的资源、服务可以在EDAS中管理。

产品集成

EDAS基于不同用户场景，以产品形式集成的云服务如下：

使用场景	集成产品	提供的功能	收费说明
基础设施管理	专有网络VPC	为集群和应用提供网络隔离，提升应用的安全性。	需要创建，独立计费。
	容器服务ACK	为EDAS提供K8s环境。	需要创建，相关资源独立计费。
	云服务器ECS	为EDAS提供ECS环境。	可以创建、导入或由EDAS代购，独立计费。
	高速通道	为混合云提供物理专线链接。	需要创建，独立计费。

使用场景	集成产品	提供的功能	收费说明
服务治理	微服务引擎MSE	为应用提供开源Nacos、Eureka和ZooKeeper注册配置中心托管，以及微服务网关。	需要创建，独立计费。
	服务网格ASM	提供多语言应用的部署能力。	无需付费。
应用运维	日志服务SLS	为应用提供文件日志。	需要开通，独立计费。
	负载均衡SLB	为应用提供负载均衡。	需要创建，独立计费。
	弹性伸缩ESS	为应用提供弹性伸缩。	产品本身免费，但所需资源，例如ECS、SLB等需要独立计费。
权限管理	访问控制RAM	为EDAS提供权限管理。	免费产品，默认开通，无需付费。
	资源管理	通过资源组辅助为EDAS提供权限管理。	免费产品，默认开通，无需付费。

组件集成

EDAS基于不同用户场景，以服务组件形式集成的云服务如下：

使用场景	集成产品	提供的功能	收费说明
服务治理	应用高可用服务AHAS	为应用提供限流降级。	需要开通，独立计费。
	性能测试PTS	为应用提供服务压测。	无需付费。
应用监控	应用实时监控服务ARMS	为应用提供监控。	• EDAS集成的功能无需付费 • 如果需要使用ARMS完整的监控能力，需要开启高级监控，独立计费。
	Prometheus监控服务	为K8s集群和多语言应用提供监控。	EDAS集成的功能无需付费。
	链路追踪Tracing Analysis	为多语言应用提供链路追踪。	EDAS集成的功能无需付费。
应用运维	分布式任务调度SchedulerX	为应用提供任务调度。	免费产品，无需付费。
	应用配置管理ACM	为应用提供配置管理。	免费产品，无需付费。
	云服务总线CSB	为应用提供环境互通和开放。	需要创建，独立计费。

7.1.2. 组件中心

7.1.2.1. 组件中心简介

组件中心围绕微服务体系，重点建设服务集成和整合能力，进而实现PaaS平台开放的生态体系。目前EDAS主要提供3类组件：微服务组件、应用诊断组件以及其他组件，您可针对自己的业务需求自行开通。

微服务组件

目前EDAS提供的微服务组件为云服务总线CSB，您可根据业务需求自行开通。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**组件中心 > 组件概览**。

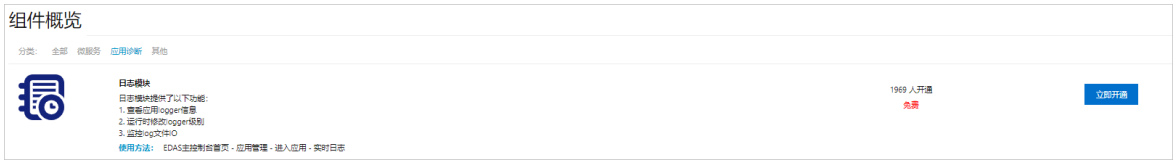
3. 在**组件概览**页面单击**微服务**页签。
4. 在**微服务**页签中**云服务总线**右侧单击**立即开通**。

开通成功后可在控制台左侧导航栏**组件中心 > 云服务总线**中管理。更多信息，请参见云服务总线CSB。

应用诊断组件

目前EDAS提供的诊断组件包括日志模块，提供对应用的在线诊断能力。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**组件中心 > 组件概览**。
3. 在**组件概览**页面选择**应用诊断**页签。
4. 在**应用诊断**页签中**日志模块**右侧单击**立即开通**。



诊断组件开通后，在组件中心还可以进行移除、查看组件版本和意见反馈等操作。

其他组件

目前EDAS提供的服务组件为分布式任务调度2.0，您可根据业务需求自行开通。关于SchedulerX的更多信息，请参见什么是分布式任务调度SchedulerX。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**组件中心 > 组件概览**。
3. 在**组件概览**页面选择**其他**页签。
4. 在**其他**页签中**分布式任务调度2.0（BETA）**右侧单击**立即开通**。

7.1.2.2. 云服务总线

7.1.2.2.1. 云服务总线CSB简介

您现在可以直接在EDAS控制台上创建、管理云服务总线CSB（Cloud Service Bus）专享实例，用于正式的服务开放场景。

管理服务开放实例

在EDAS控制台上可以创建云服务总线（CSB）专享实例，也可以对这些实例进行扩容、缩容操作。一个专享实例就如同一个“应用”网关，可以用来管理和控制目标环境内应用对外的服务开放，也可以引入外部服务并进行管理控制。

在创建专享实例时，需要您选择该实例所服务的目标环境，包括VPC、交换机、安全组，以及用于访问该实例的统一入口SLB。这种专享实例不需要您提供自己VPC内的ECS来部署。

您仍可以选择之前的方式，即在原有CSB控制台上申请新的专享实例，这需要您提供自己VPC内的ECS并联系产品服务团队来部署和激活。这种方式在一些特殊的，如复杂混合云联动的场景才有必要。

通常情况下，建议您采用新的方式，在EDAS控制台上进行云服务总线CSB实例的创建和管理。

管理服务开放实例的具体操作，请参见管理服务开放实例。

7.1.2.2.2. 管理服务开放实例

每一组CSB节点（Broker）集群被视为一个独立的CSB实例，通常负责一个业务域内能力的对外开放。您想要开放应用中的HSF服务，需要先在EDAS中创建用于开放服务的CSB实例。您也可以对这些实例进行管理，以保证HSF服务可以正常开放。

前提条件

请确保已经创建了用于创建CSB实例的VPC、交换机、安全组，并在该VPC内创建了SLB。

SLB的监听端口和探活地址会根据下表自动配置。

后端协议/端口	会话保持	调度算法	探活协议/端口	探活地址	说明
TCP: 8086	关闭	加权最小连接数	HTTP: 8086	/monitor/status .1688	CSB RESTFul接入
TCP: 9081	关闭	加权最小连接数	HTTP: 8086	/monitor/status .1688	CSB WebService 接入
TCP: 8081	关闭	加权最小连接数	HTTP: 8086	/monitor/status .1688	CSB结点互联

> **说明**
>
> - 每个用户最多创建5个CSB实例，如有特殊需求，请联系CSB技术支持人员。
> - 仅支持VPC内的非共享型私网SLB，目前支持ecs.c5.xlarge 4C8G或ecs.c5.large 2C4G规格的ECS。正式环境建议使用ecs.c5.xlarge 4C8G。
> - 如果需要公网地址，请将SLB绑定弹性公网IP。
> - CSB实例创建后，实例名、VPC、交换机、安全组、SLB、ECS规格均不可修改。
> - CSB实例创建后，不可删除SLB，否则会导致CSB服务不可用。

创建实例

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**组件中心 > 云服务总线**。
3. 在**实例列表**页面上方选择地域，单击**新建实例**。
4. （可选）在弹出的**SLB操作授权**对话框中单击**确认**，跳转到**云资源访问授权**页面，单击**同意授权**，授权CSB绑定和解绑SLB。
5. 返回**新建实例**页面，设置实例参数，然后单击**确认**。
   - 实例名的命名格式必须以缺省前缀 `csb_aliyun_<region>_` 开始，后面为5~64位字符的字母、数字及或下划线。其中 `<region>` 是当前地区名，例如*cn_hangzhou*。
   - 系统会根据您账户下的VPC、交换机、安全组、SLB、ECS规格及部署ECS数量等为您成默认配置。您也可以根据实际需要在各参数下拉菜单中选择对应资源。

   实例创建完成后，会显示在实例列表中。

扩容实例

1. 登录EDAS控制台。

2. 在左侧导航栏中选择**组件中心 > 云服务总线**。
3. 在**实例列表**页面上方选择地域，然后在需要扩容的实例的**操作**列单击**节点管理**。
4. 在**实例部署管理**对话框中单击**扩容节点**右侧的图标，选择扩容的节点数量。

5. 在弹出的确认对话框中单击**确认**。
   选择完成后，系统会自动为您申请同规格的ECS实例，并显示即时状态。

缩容实例

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**组件中心 > 云服务总线**。
3. 在**实例列表**页面上方选择地域，然后在需要扩容的实例的**操作**列单击**节点管理**。
4. 在**实例部署管理**对话框中目标实例的**操作**列单击**释放**。
5. 在弹出的确认对话框中单击**确认**。

暂停和删除实例

当不再需要使用CSB实例时，可以暂停和删除实例。

- 暂停实例：实例暂停后，将立即释放实例的所有节点资源及实例上部署的应用数据，三个月内可以恢复继续使用。
- 删除实例：实例删除后，将立即释放、删除实例的所有节点资源及实例上部署的应用数据。

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**组件中心 > 云服务总线**。
3. 在**实例列表**页面上方选择地域，然后在需要暂停或删除的实例的**操作**列单击**更多**右侧的图标，然后在列表中单击**实例暂停**或**实例删除**。
4. 在弹出的确认对话框中单击**确认**。

相关操作

在CSB实例列表中单击具体实例名，可以跳转到CSB控制台，进行服务发布、订阅等操作。具体操作，请参见CSB中开放平台相关章节。

7.1.2.3. 批量运维

在EDAS控制台中，可以使用机器指令对安装了Agent的ECS实例进行批量运维操作。可以按集群、应用和实例批量执行命令，解决多个实例重复运维的烦恼。

适用场景

批量运维可以解决以下场景的运维问题：

- 场景一：多个实例或者集群执行同一操作。
- 场景二：同时执行多个命令任务。
- 场景三：查询追溯任务执行历史。

执行批量运维命令

1. 登录EDAS控制台。
2. 在左侧导航栏中选择**组件中心 > 批量运维**。
3. 在**批量运维**页面选择地域（Region）和命名空间。
4. 单击**按集群**、**按应用**或**按实例**页签，以确定执行维度。

   本文档以**按集群**维度为例说明，另外两种方式的操作和集群基本一致。
5. 在**选择集群**右侧单击**新增**，然后在**选择集群**对话框中的左侧区域勾选具体集群（或通过名称关键字搜索），然后单击 > 按钮将集群添加到右侧**已选择**区域，再单击**确定**。
6. 在**命令**右侧的文本框中输入命令。
7. （可选）如果您选择的集群全部是同一类型的集群，则无需选择执行范围。如果您的选择集群包含两种及以上类型的集群，则需要选择执行范围：
   - **在ECS实例上执行**。
   - **在Docker容器执行**。
   - 既在ECS实例上执行，也在Docker容器执行（都勾选）。

   系统使用admin账号登录机器执行命令。
8. 单击**执行**。

查看执行结果及详情

在命令执行后，在**提示**窗口单击**确定**，控制台会自动生成**执行结果**页签。

执行结果区域显示详情列表，包括此次批量运维的各个ECS和Docker实例的**应用类型**、**IP地址**、**VPC ID**、**状态**和命令执行详情。

执行详情区域会显示命令在实例上的详细执行过程。如果失败，会提示导致执行失败的原因。

> **说明** 不同命令执行时间各不相同，如果尚未显示执行结果或执行结果显示不全，请单击**刷新**按钮。

查看执行记录

1. 在**执行记录**页面下方可以查看批量运维操作记录，包括操作人、创建时间、结束时间、执行命令、（执行结果）状态。
   - 如果是当前账号为主账号，可以看到主账号及该主账号下所有子账号执行的所有批量运维命令。
   - 如果是当前账号为子账号，则只能看到该子账号执行的批量运维命令。

   操作记录按照时间倒序排列，同时支持按照操作人和创建或结束时间进行筛选。
2. 在详情列单击**查看**可以跳转到详情页面。

7.2. 权限管理

7.2.1. 权限管理概述

您在EDAS上托管的应用可能包含多个服务或子系统，这些服务或子系统又可能由不同团队、成员进行开发、运维。EDAS通过账号体系及基于账号体系的一系列权限管理操作，提供企业级的权限管理系统，帮助您对应用、资源和数据进行必要的隔离和权限控制，以保证其安全性。

EDAS内置权限管理和RAM权限管理

EDAS内置了一套权限管理系统，又接入了阿里云访问控制RAM（Resource Access Management）的权限管理。

- RAM权限管理操作，请参见RAM简介。
- EDAS内置权限管理操作，请参见管理EDAS内置权限（不推荐）。

为了能够统一管理EDAS和其他阿里云产品的权限，推荐您使用RAM权限管理。EDAS也提供了将内置的权限管理迁移至RAM权限管理的方案。具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。

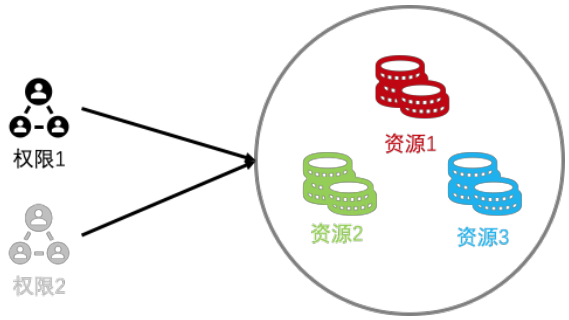
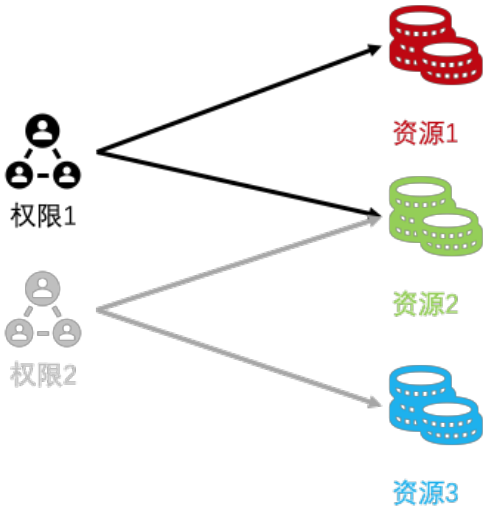
> **说明** EDAS内置权限管理模式下，阿里云账号如果要对子账号进行权限管理，需要登录EDAS控制台，为子账号分配EDAS的权限和资源。迁移至RAM后，阿里云账号在EDAS控制台无法再为子账号授权，需要登录到RAM控制台为子账号授予EDAS的相关权限。

EDAS目前处于EDAS内置权限管理与RAM权限管理共存的过渡状态，目前您的RAM用户或子账号所使用的权限管理规则如下：

- 未使用过EDAS内置权限管理的子账号，直接使用RAM权限管理，不能再使用EDAS内置权限管理。
- 使用过EDAS内置权限管理的子账号：
  - 拥有RAM中AliyunEDASFullAccess权限的子账号，直接使用RAM鉴权，不再开放EDAS内置鉴权。
  - 使用过EDAS内置权限管理的子账号，建议手动切换为RAM鉴权，具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。如果不做任何改动，可继续使用EDAS内置鉴权。

为什么要使用RAM权限管理

RAM是阿里云提供的资源访问控制服务。通过设置权限策略（Policy），您可以集中管理您的用户（例如员工、系统或应用程序），以及控制用户可以访问哪些资源，例如限制您的用户只拥有对某一个EDAS应用的读权限。

EDAS内置权限管理	RAM权限管理
	
阿里云账号为子账号分配资源后，所有的权限都会匹配这些资源。例如，当授予子账号APP1和APP2的资源后，如果再授予子账号部署应用和停止应用的权限，那么，该子账号会有权限对APP1和APP2进行部署和停止操作。资源粒度较为粗糙，不能更精确地控制权限。	相比EDAS内置的权限管理，RAM的权限控制更为精细。每条权限都可以配置所拥有的资源。例如，阿里云账号可以授予RAM用户部署App和停止App的权限，同时，部署App的权限配置上资源APP1和APP2，停止App的权限配置上资源APP2和APP3。此时，RAM用户可以部署APP1和APP2，不能部署APP3；可以停止APP2和APP3，不能停止APP1。

相比EDAS内置权限管理，RAM权限管理的语法更为丰富，不仅支持用通配符进行模糊匹配，还支持多种鉴权策略组合，实现更复杂的鉴权逻辑。

以下面的RAM策略为例：

```
{
    "Statement": [
      {
        "Action": [
          "edas:ReadApplication"
        ],
        "Effect": "Allow",
        "Resource": ["acs:edas:*:*:namespace/*/application/*"]
      },
      {
        "Action": [
          "edas:ReadApplication"
        ],
        "Effect": "Deny",
        "Resource": ["acs:edas:cn-beijing:*:namespace/*/application/12345678"]
      }
    ],
    "Version": "1"
}
```

该RAM策略有2条语句：

- 第1条是允许语句，权限动作**edas:ReadApplication**代表查看APP，针对的资源是通配符*，代表所有APP。第1条语句的意思就是授予RAM用户查看EDAS中所有的App的权限。
- 第2条是禁止语句，资源是ID为*12345678*的App，代表的意思为禁止查看ID为*12345678*的App。

以上两条语句相结合，则代表：授予RAM用户查看除ID为*12345678*的App之外的其它所有App的权限。

RAM权限管理语法还有条件表达式等多种功能。更多信息，请参见权限策略概览。

更细粒度的鉴权

更强大的语法

EDAS内置权限和RAM权限的映射关系

EDAS内置权限与RAM权限并不完全匹配，因此是无法等效替换的。EDAS提供了权限策略转换功能，可将EDAS内置权限转换为尽量一致的RAM权限。其中，子账号拥有的EDAS的资源（应用和集群）默认会赋给RAM的所有权限点使用。

权限映射对照表

EDAS内置权限	RAM权限	RAM资源
Super Admin(All privileges)	edas:*	acs:edas:*:*:*
代理主账号	edas:ManageSystem	acs:edas:*:*:*
系统管理-操作日志	edas:ReadOperationLog	acs:edas:*:*:*
应用管理-编辑微服务空间	edas:ManageNamespace	acs:edas:*:*:namespace/${namespaceId}
应用管理-查询微服务空间	edas:ReadNamespace	acs:edas:*:*:namespace/${namespaceId}
资源管理-创建集群	edas:CreateCluster	acs:edas:*:*:namespace/*

EDAS内置权限	RAM权限	RAM资源
资源管理-查看集群	edas:ReadCluster	acs:edas:*:*:namespace/*/cluster/${clusterId}
资源管理-管理和删除集群	edas:ReadCluster edas:ManageCluster	acs:edas:*:*:namespace/*/cluster/${clusterId}
应用管理-创建应用	edas:CreateApplication	acs:edas:*:*:namespace/*
应用管理-部署、启动、扩容和删除应用	edas:ManageApplication edas:ReadApplication	acs:edas:*:*:namespace/*/application/${applicationId}
应用管理-查看应用信息	edas:ReadApplication	acs:edas:*:*:namespace/*/application/${applicationId}
应用管理-配置容器和编辑应用JVM参数	edas:ConfigApplication edas:ReadApplication	acs:edas:*:*:namespace/*/application/${applicationId}
应用管理-设置日志路径	edas:ManageAppLog edas:ReadApplication	acs:edas:*:*:namespace/*/application/${applicationId}
ECS代购	edas:ECSPurchase	acs:edas:*:*:*
SLB代购	edas:SLBPurchase	acs:edas:*:*:*
SLS代购	edas:SLSPurchase	acs:edas:*:*:*

7.2.2. 账号体系

为了更好地保护企业信息安全，实现企业级的账号管理，EDAS自身提供了一套账号体系，同时还接入了阿里云访问控制RAM（Resource Access Management）的账号体系。EDAS已逐渐从EDAS内置账号体系迁移至RAM的账号体系。

相关概念

账号体系中包含阿里云账号、RAM用户、子账号（EDAS内置，不推荐）和角色等概念。

- 阿里云账号
  EDAS中，阿里云账号拥有其下所有资源和EDAS的所有操作的权限。购买EDAS的阿里云账号也是付费账号。
  在EDAS控制台的**系统管理 > 主账号**菜单内，可以查看当前阿里云账号的**应用实例数限额**、**实际应用实例数**和**产品系列**。

  > **说明** EDAS原有付费阿里云账号可以绑定其他未开通EDAS的阿里云账号，如需解绑原有付费账号绑定的其他阿里云账号，请提交工单。

- RAM用户
  EDAS系统支持RAM的账号体系。在使用EDAS时，推荐使用RAM的账号体系。EDAS的阿里云账号可以登录RAM控制台，创建RAM用户，按需为RAM用户分配最小权限，实现各司其职的高效企业管理。
  在EDAS控制台的**系统管理 > 子账号**菜单内，可以查看和完成以下操作：
  - 阿里云账号登录控制台可以查看阿里云账号对应的所有子账号列表。
  - 可以在页面右上角单击**同步子账号**完成RAM子账号的同步。
  - 配置过EDAS内置权限并且未迁移至RAM授权的子账号可以管理角色、授权应用和资源组。
  - 可以迁移账号权限至RAM，详情请参见将EDAS内置的权限管理切换为RAM权限管理。

- 子账号（EDAS内置，不推荐）
  EDAS原有账号体系中提供了EDAS独立的子账号，现已不支持新建，并且推荐您将子账号切换到RAM。具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。

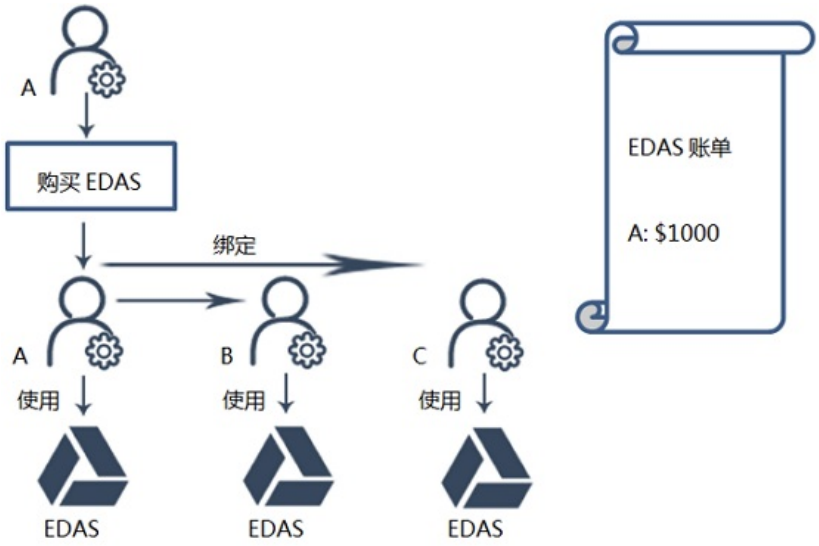
> **说明** 在EDAS内置的子账号未切换到RAM前，您还可以对EDAS子账号进行权限管理。具体操作，请参见管理EDAS内置权限（不推荐）。

- 角色
  角色即拥有一系列指定权限的虚拟用户，没有确定的身份认证密钥，需要被一个守信的实体用户扮演才能正常试用。
  EDAS中可以创建角色，也可以使用RAM的角色。
- 权限策略
  权限策略是用语法结构描述的一组权限的集合，可以精确地描述被授权的资源集、操作集以及授权条件。
  权限策略只能在RAM中创建。EDAS内置的权限控制仅支持对子账号授权应用或资源组。

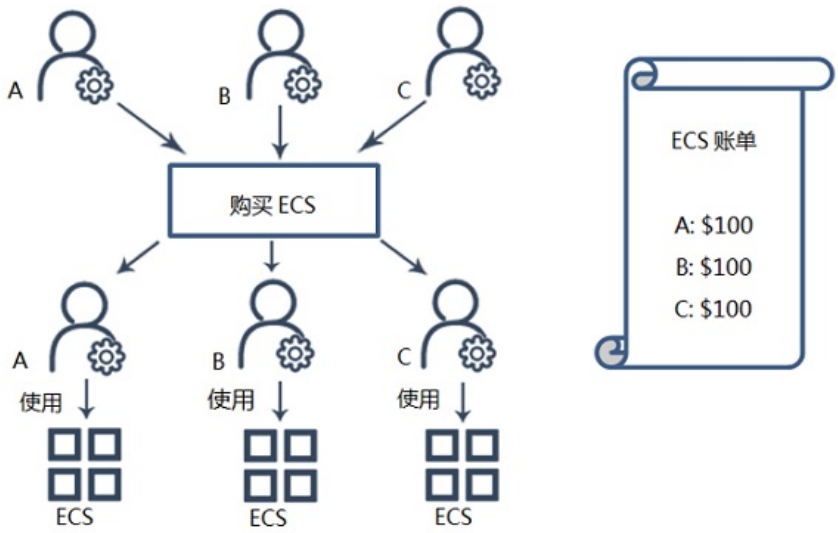
使用场景

下文以几个典型场景为例来进一步说明EDAS账号体系的使用场景。

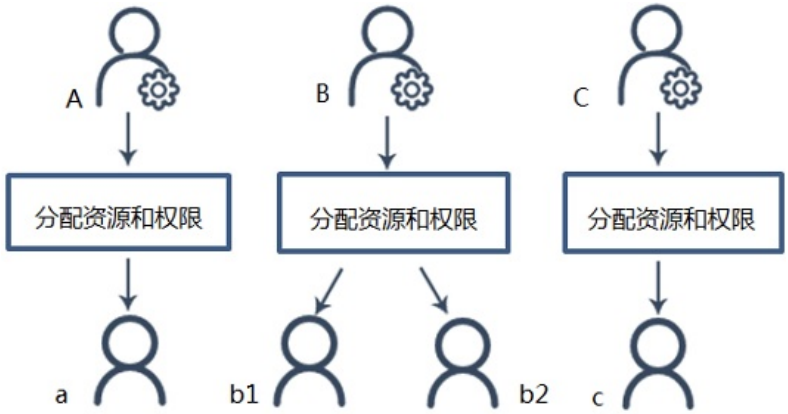
- 场景一
  某公司用账号A购买了EDAS，那么A就是一个付费账号，同时也是一个阿里云账号。该公司还有其他两个部门都需要使用EDAS，于是可以在阿里云账号A下新建两个部门的子账号或RAM用户B、C并授予EDAS的管理权限。那么账号B、C不需要付费购买EDAS就可以正常使用EDAS。



- 场景二
  假设子云账号B、C需要使用EDAS的完整功能，如创建应用、运行应用等，则必须自行购买ECS等资源，而不能用账号云账号A进行购买。



- 场景三
  准备好资源后，三个阿里云账号对应的部门分别创建了各自的子账号或RAM用户，用于权限及资源的具体分配和管理。
  - 账号A给子账号或RAM用户a授予了所有的ECS资源和所有权限。
  - 账号B创建了应用管理员和运维管理员的两个角色，并将这两个角色分别授予给了子账号或RAM用户b1，b2。
  - 账号C创建了一个查看应用的角色，授权给了c。



7.2.3. 将EDAS内置的权限管理切换为RAM权限管理

为了能够用统一的账号体系来管理阿里云产品（包含EDAS）的权限，EDAS已将内置的权限管理迁移至RAM。本文介绍如何将EDAS内置的权限管理切换为RAM的权限管理，以及如何在RAM控制台为RAM用户授予EDAS的权限策略。

背景信息

权限策略语言的基本结构和语法，请参见权限策略语法和结构。

参数	描述
效力（Effect）	授权效力包括两种：允许（Allow）和拒绝（Deny）。

参数	描述
操作（Action）	操作是指对具体资源的操作。操作支持多值，取值为：所定义的云服务的操作名称。格式为：`<service-name>:<action-name>`。 • `service-name`：阿里云产品名称。 • `action-name`：相关的云服务的操作名称。
资源（Resource）	资源是指被授权的具体对象。格式遵循阿里云ARN（Aliyun Resource Name）的统一规范：`acs:<service-name>:<region>:<account-id>:<relative-id>`。
限制条件（Condition）（可选）	限制条件是指授权生效的限制条件。限制条件由一个或多个条件子句构成。一个条件子句由条件关键字（Key）、条件操作类型（Operator）和条件值组成（Value）。

步骤一：生成EDAS的权限策略

您可以参见以下三种方式来查看或生成EDAS的权限策略。

无论您使用了RAM权限还是EDAS内置权限，都可以在权限策略示例库中来查找您想授予的权限策略。更多信息，请参见权限策略示例库。

本文仅介绍简单的场景下如何操作。更多信息，请参见使用EDAS权限助手生成权限策略。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 权限助手**。
3. 在**权限助手**页面单击**创建权限策略**。
4. 在**创建权限策略**配置向导中设置权限策略参数。
   i. 在**创建自定义权限策略**页签配置权限策略参数，然后单击**下一步**。

参数	描述
策略名称	自定义的策略名称。
备注	权限策略的备注信息。
新增权限语句	a. 单击**新增权限语句**。 b. 在**加授权语句**面板设置**权限效力**和**操作与资源授权**，然后单击**确认**。 **注意** 在添加权限策略时，同时只能选择**允许**或**拒绝**中的一种权限效力。 c. 在**创建自定义权限策略**页签内的权限策略列表的**操作**列根据需求克隆、编辑或删除权限。

   ii. 在**策略预览**页签预览权限策略，并在权限策略文本框的右上角单击**复制**，然后在面板下方单击**完成**。
   控制台面板将会提示`新增策略授权成功`，单击**返回列表查看**可查看和管理新建的权限策略。
5. 在**授权策略**区域复制生成的授权策略。

如果您已经使用EDAS内置权限管理完成了账号授权，可以在EDAS控制台直接将配置的权限转换为RAM权限策略。

1. 登录EDAS控制台。

2. 在左侧导航栏选择**系统管理 > 子账号**。
3. 在**子账号**页面选择添加了EDAS内置权限的子账号，在**RAM鉴权**列单击**生成RAM权限策略**。
4. 在弹出的**RAM权限策略**对话框复制权限策略，然后单击**确定**。

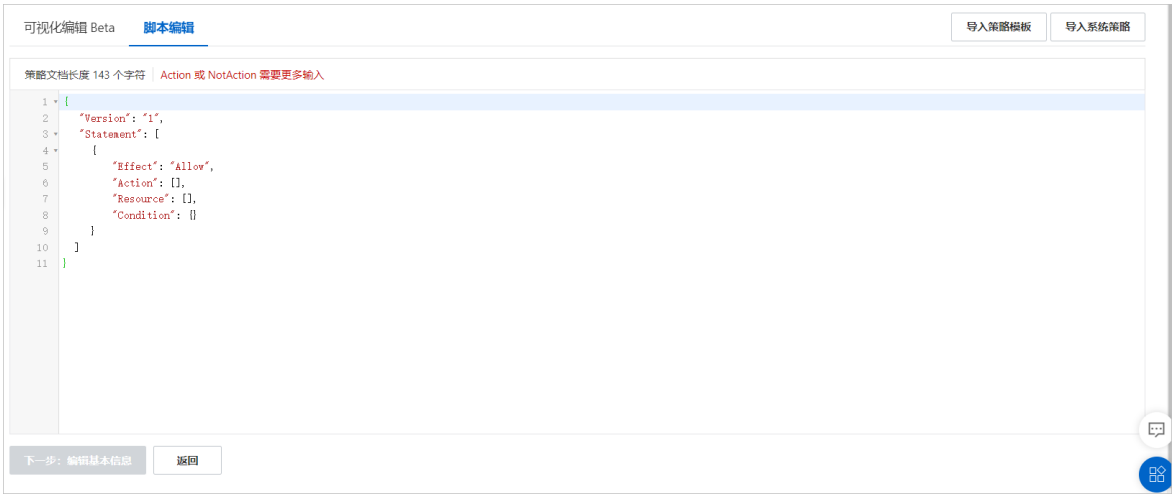
方式一：查看权限策略示例库

方式二：使用权限助手生成权限策略

方式三：直接转换EDAS内置权限为RAM权限策略

步骤二：创建权限策略

1. 登录RAM控制台。
2.
3. 在**权限策略管理**页面单击**创建权限策略**。
4. 在**新建自定义权限策略**页面设置策略参数，完成设置后单击**确定**。



参数	描述
策略名称	自定义的策略名称，策略名称仅可包含大小写字母、数字以及短划线（-）。

参数	描述
备注（可选）	权限策略的备注信息。
配置模式	单击**脚本配置**，在**策略内容**文本框内输入步骤一：生成EDAS的权限策略中生成的权限策略。

步骤三：创建RAM用户并添加授权

1. 登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 用户**。
3. 在**用户**页面，单击**创建用户**。
4. 在**创建用户**页面的**用户账号信息**区域，输入**登录名称**和**显示名称**。

   > **说明** 单击**添加用户**，可一次性创建多个RAM用户。

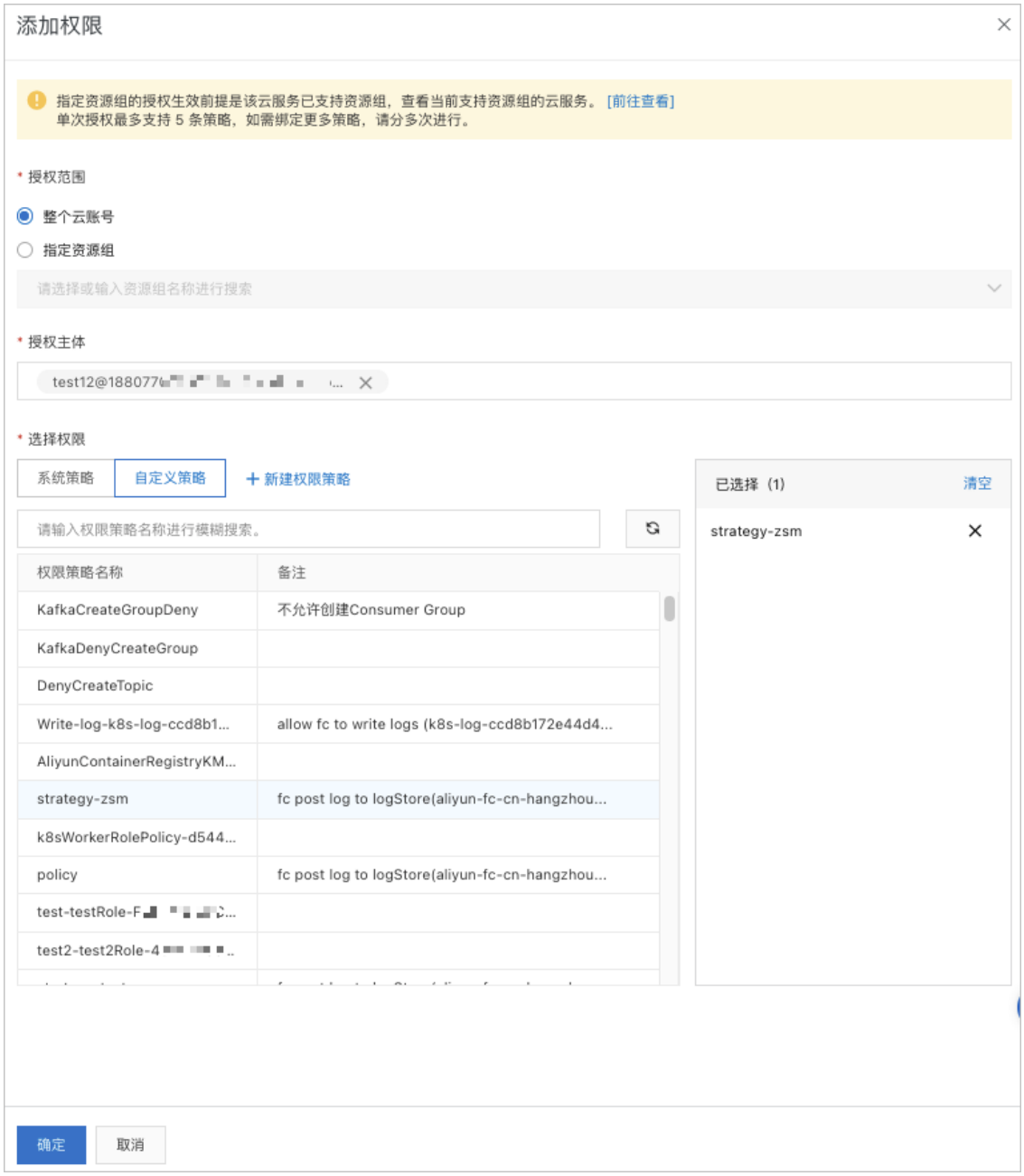
5. 在**访问方式**区域，选择访问方式。
   - **控制台访问**：设置控制台登录密码、重置密码策略和多因素认证策略。

     > **说明** 自定义登录密码时，密码必须满足密码复杂度规则。关于如何设置密码复杂度规则，请参见设置RAM用户密码强度。

   - **OpenAPI调用访问**：自动为RAM用户生成访问密钥（AccessKey），支持通过API或其他开发工具访问阿里云。

     > **说明** 为了保障账号安全，建议仅为RAM用户选择一种登录方式，避免RAM用户离开组织后仍可以通过访问密钥访问阿里云资源。

6. 单击**确定**。
7. 返回**用户**页面，在用户列表中目标RAM用户的**操作**列单击**添加权限**。
8. 在**添加权限**面板内设置权限，然后单击**确定**。

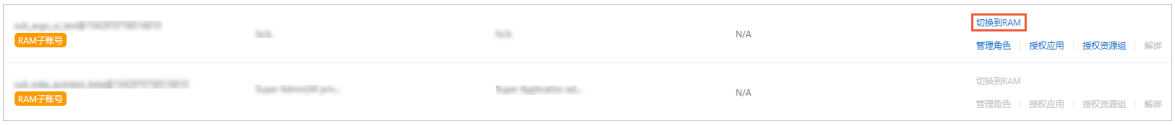


参数	描述
授权范围	包含**整个云账号**和**指定资源组**。请根据实际需求选择授权范围。
授权主体	会自动填入当前RAM用户的登录名称。
选择权限	选择**自定义策略**，然后在搜索框内搜索设置的权限策略名称，在**权限策略名称**区域单击搜索出来的策略名称，即可选中策略。

步骤四：在EDAS控制台将授权切换为RAM授权

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 子账号**。

3. 在**子账号**页面目标子账号的**RAM鉴权**列单击**切换到RAM**。



> **说明**
> - 已经使用了RAM授权的子账号，**操作**列的按钮将会无法单击。
> - 仍使用EDAS内置授权的子账号，可以选择切换到RAM授权，切换后将无法重新使用EDAS内置授权。

切换到RAM授权时，EDAS会预先判断该子账号是否已经在RAM控制台授予了EDAS的权限。

- 如果已经完成了RAM授权，在弹窗中单击**确定**，即可成功切换为RAM鉴权。
- 若子账号没有在RAM控制台被授予EDAS权限，将提示前往RAM控制台完成授权。

相关文档

- 权限管理概述
- 权限策略示例库

7.2.4. RAM权限管理

7.2.4.1. RAM简介

访问控制RAM（Resource Access Management）是阿里云提供的一项管理用户身份与资源访问权限的服务。通过RAM，您可以创建、管理RAM用户（例如员工、系统或应用程序），并可以控制这些RAM用户的权限。
当您的企业存在多用户协同操作时，使用RAM可以让您避免与其他用户共享云账号密钥，按需为用户分配最小权限，从而降低企业信息安全风险。

> **说明** EDAS中仅介绍和EDAS相关的RAM操作。RAM本身对RAM用户、角色、权限策略的更多信息，请参见什么是访问控制。

7.2.4.2. 管理RAM用户

EDAS系统支持阿里云访问控制RAM（Resource Access Management）的账号体系，云账号可以通过创建RAM用户，避免与其他用户共享账号密钥，按需为RAM用户分配最小权限，实现各司其职的高效企业管理。

RAM用户简介

云账号在使用EDAS过程中，通常需要不同身份的用户来做不同类型的工作，如应用管理员（创建应用、启动应用、查询应用和删除应用等）、运维管理员（查看资源、查看应用监控、管理报警规则、管理限流降级规则等）等。云账号可以通过给RAM用户分配不同的角色和资源，来实现分权分责。这种权限管理模式，类似于Linux系统中的系统用户和普通用户，系统用户可以对普通用户进行权限管理。

RAM用户说明：

- RAM用户由云账号在RAM控制台中创建，不需要校验合法性，名字在云账号内唯一即可。
- 不同于一般的阿里云账号登录，RAM用户有独立的登录入口：https://signin.aliyun.com。

使用RAM用户登录EDAS控制台

1. 使用云账号登录RAM控制台。
2. 在**概览**页面的**账号管理**区域单击**用户登录地址**的URL。

> **说明** 不同的主账号创建的RAM账号的登录链接URL是不同的。

3. 在**RAM用户登录**页面输入RAM用户的登录名称，单击**下一步**。
4. 输入登录密码，然后单击**登录**。

> **说明** 如果云账号在创建RAM用户时选择了**用户在下次登录时必须重置密码**，那么RAM用户在第一次登录控制台后，会被要求重置密码。

5. 使用RAM用户登录EDAS控制台。

移除RAM用户权限

1. 使用云账号登录RAM控制台。
2. 在左侧导航栏选择**身份管理 > 用户**。
3. 在**用户**页面单击选择需要解除授权的用户登录名称。
4. 在**用户基本信息**页面单击**权限管理**页签。
5. 在**权限管理**页签中目标权限的**操作**列单击**移除权限**，然后在弹窗中单击**确定**。

删除RAM用户

1. 使用云账号登录RAM控制台。
2. 在左侧导航栏选择**身份管理 > 用户**。
3. 在**用户**页面要删除RAM用户的**操作**列单击**删除**。
4. 在**删除用户**对话框中查看删除用户的影响，当确认要删除时单击**我已知晓风险，确认删除**。

7.2.4.3. 借助RAM角色实现跨云账号访问资源

使用企业A的阿里云账号（主账号）创建RAM角色并为该角色授权，并将该角色赋予企业B，即可实现使用企业B的主账号或其RAM用户（子账号）访问企业A的阿里云资源的目的。

背景信息

假设企业A购买了多种云资源来开展业务，并需要授权企业B代为开展部分业务，则可以利用RAM角色来实现此目的。RAM角色是一种虚拟用户，没有确定的身份认证密钥，需要被一个受信的实体用户扮演才能正常使用。为了满足企业A的需求，可以按照以下流程操作：

1. 企业A创建RAM角色
2. 企业A为该RAM角色添加AliyunEDASFullAccess权限
3. 企业B创建RAM用户
4. 企业B为RAM用户添加AliyunSTSAssumeRoleAccess权限
5. 企业B的RAM用户通过控制台或API访问企业A的资源

可以为RAM角色添加的EDAS系统权限策略包括：AliyunEDASFullAccess，EDAS的完整权限。

> **说明**
> - 被访问的角色需要具有AliyunEDASFullAccess权限。
> - RAM子用户只能扮演非自己主账户的角色，即子账户不能扮演子账户所在的主账户下的角色，只能扮演其它主账户下的拥用AliyunEDASFullAccess权限的角色。用户在自己扮演自己主账户角色的情况下将无法访问EDAS，请退出角色登录再访问EDAS。
> - 子账户扮演具有AliyunEDASFullAccess权限的角色成功后，即拥有所扮演主账号的所有EDAS权限。

步骤一：企业A创建RAM角色

首先需要使用企业A的阿里云账号（主账号）登录RAM控制台并创建RAM角色。

1. 登录RAM控制台，在左侧导航栏中单击**身份管理 > 角色**，并在页面上单击**创建角色**。

2. 在**创建角色**配置向导中执行以下操作并单击**完成**。
   i. 在**选择类型**页签的**选择可信实体类型**区域中选择**阿里云账号**，并单击**下一步**。
   ii. 在**配置角色**页签的**角色名称**文本框内输入RAM角色名称。

   > **说明** RAM角色名称中允许使用英文字母、数字和“-”，长度不超过64个字符。

   iii. 在**配置角色**页签的**选择云账号**区域中选择**其他云账号**，并在文本框内输入企业B的云账号。

步骤二：企业A为该RAM角色添加权限

新创建的角色没有任何权限，因此企业A必须为该角色添加权限。

1. 在RAM控制台左侧导航栏中单击**身份管理 > 角色**。
2. 在页面上单击目标角色**操作**列中的**添加权限**。
3. 在**添加权限**面板中设置**授权范围**，在**选择权限**区域中单击**自定义策略**，通过关键字搜索需要添加的权限策略，并单击权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。

   > **说明** 可添加的权限参见背景信息部分。

4. 在**添加权限**的**授权结果**页面上，查看授权信息摘要，并单击**完成**。

步骤三：企业B创建RAM用户

接下来要使用企业B的阿里云账号（主账号）登录RAM控制台并创建RAM用户。

1. 登录RAM控制台，在左侧导航栏中选择**身份管理 > 用户**，并在**用户**页面上单击**创建用户**。
2. 在**创建用户**页面的**用户账号信息**区域中，输入**登录名称**和**显示名称**。

   > **说明** 登录名称中允许使用英文字母、数字、“.”、“_”和“-”，长度不超过128个字符。显示名称不可超过24个字符或汉字。

3. （可选）如需一次创建多个用户，则单击**添加用户**，并重复上一步。
4. 在**访问方式**区域，选中**控制台访问**或**Open API 调用访问**，并单击**确定**。

   > **说明** 为提高安全性，请仅选中一种访问方式。

   - 如果选中**控制台访问**，则完成进一步设置，包括自动生成默认密码或自定义登录密码、登录时是否要求重置密码，以及是否开启MFA（多因素认证）。
   - 如果选中**Open API 调用访问**，则RAM会自动为RAM用户创建AccessKey（API访问密钥）。

     > **注意** 出于安全考虑，RAM控制台只在创建AccessKey时提供一次查看或下载AccessKeySecret的机会，因此请务必将AccessKeySecret记录到安全的地方。

5. 在**手机验证**对话框中单击**获取验证码**，并输入收到的手机验证码，然后单击**确定**。
   创建的RAM用户显示在**用户**页面上。

步骤四：企业B为RAM用户添加权限

企业B必须为其主账号下的RAM用户添加AliyunSTSAssumeRoleAccess权限，RAM用户才能扮演企业A创建的RAM角色。

1. 在RAM控制台左侧导航栏中选择**身份管理 > 用户**。
2. 在**用户**页面上找到需要授权的用户，单击**操作**列中的**添加权限**。
3. 在**添加权限**面板设置**授权范围**，在**选择权限**区域中通过关键字搜索AliyunSTSAssumeRoleAccess权限策略，并单击该权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。

4. 在**添加权限**的**授权结果**页面上，查看授权信息摘要，并单击**完成**。

后续步骤

完成上述操作后，企业B的RAM用户即可按照以下步骤登录控制台访问企业A的云资源或调用API。

- 登录控制台访问企业A的云资源
  i. 在浏览器中打开RAM用户登录入口。
  ii. 在**RAM用户登录**页面上，输入RAM用户登录名称，单击**下一步**，并输入RAM用户密码，然后单击**登录**。

  > **说明** RAM用户登录名称的格式为 <子用户名称>@<默认域名>，或<子用户名称>@<企业别名>，例如username@company-alias.onaliyun.com或username@company-alias。

  iii. 在控制台页面上，将光标移到右上角头像，并在浮层中单击**切换身份**。
  iv. 在**阿里云 - 角色切换**页面，输入企业A的**企业别名**或**默认域名**或**UID**，以及**角色名**，然后单击**切换**。
  v. 对企业A的阿里云资源执行操作。
- 使用企业B的RAM用户通过API访问企业A的云资源
  要使用企业B的RAM用户通过API访问企业A的云资源，必须在代码中提供RAM用户的AccessKey ID、AccessKey Secret和Security Token（临时安全令牌）。

7.2.4.4. 借助资源组进行权限管理

EDAS支持阿里云资源管理的资源组功能。EDAS接入资源组服务后，将EDAS的集群和应用分派给不同的资源组，并为RAM用户配置资源组权限，可以简化权限配置，方便企业管理。

前提条件

- 非RAM用户（EDAS原有子账号）需要将EDAS自有的权限管理切换为RAM权限管理模式。具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。
- 云账号（主账号）和RAM用户自动开启资源管理，无需操作。

背景信息

阿里云资源管理服务包含一系列支持企业IT治理的资源管理产品集合。通过资源管理服务，您可以按照业务需要搭建合适的资源组织关系，使用资源组，分层次地组织和管理EDAS中的集群和应用等资源。

资源组（Resource Group）是在阿里云账号下进行资源分组管理的一种机制，资源组能够帮助您解决单个阿里云账号内的资源分组和授权管理的复杂性问题。

- 对单个阿里云账号下多个地域、多种云资源进行集中的分组管理。
- 为每个资源组设置管理员，资源组管理员可以独立管理资源组内的所有资源。

EDAS自有权限管理和RAM权限管理的关系

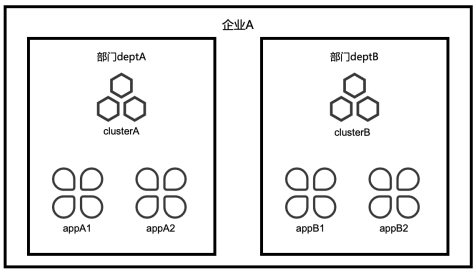
EDAS包含以下两种相互独立的权限管理模式。

> **说明** 一个阿里云的账号，只能同时使用一种权限管理模式，推荐使用RAM权限管理模式。

- 接入RAM前，EDAS通过自己的账号体系和资源组，独立对子账号进行权限管理。
- 接入RAM后，EDAS借助资源管理中的资源组，对RAM用户进行权限管理。

为什么要借助资源组进行权限管理

以一个示例场景说明为什么要借助资源组进行权限管理。



企业A有两个部门，deptA和deptB。部门deptA使用RAM用户subA管理集群clusterA和应用appA1和appA2；部门deptB使用RAM用户subB管理集群clusterB和应用appB1和appB2。

在RAM中有两种权限管理方式，直接使用RAM进行权限管理和借助资源组进行权限管理。

直接使用RAM和借助资源组进行权限管理对比

对比	直接使用RAM权限管理	借助资源组进行权限管理
使用场景	适用于细粒度的权限管理场景	适用于粗粒度的权限管理场景
原理	使用该企业的云账号在RAM控制台中分别为部门deptA和deptB创建RAM用户subA和subB，并为subA和subB创建权限策略policyA和policyB，在策略中分别配置集群、应用资源，以及操作权限。	使用该企业的云账号在资源管理控制台创建资源组groupA和groupB，将部门deptA和deptB的集群和应用资源分别添加到两个资源组中。在RAM控制台为两个部门创建RAM用户subA和subB，并创建通用的自定义策略。该策略可以在两个资源组内对不同的资源和子账号分别生效，互不影响。
特点	这种方式有两个缺点： • 配置重复 subA和subB是一样的权限控制需求，仅仅是管理的资源不同，但需要创建两个策略，分别配给两个RAM用户。 • 改动频繁 每次创建或删除资源，例如subA新建了应用appA3、appA4，就需要给在policyA中添加对应的AppId配置。	这种方式的优势： • 简化权限配置 资源组帮助管控资源范围，因此授权的时候，可以粗粒度的调整资源，不需要再精细到单个应用或集群。 • 复用权限策略 您不再需要因为资源不同，为每个不同的子账号都配置一份策略，策略只要关注action即可。

> **说明** 两种权限管理方式使用场景不同，需要根据具体需求选择。

使用限制

资源组能够简化EDAS权限配置，但在使用资源组时，需要注意以下限制。

- 微服务空间不在资源组的管理范围内。资源组能够覆盖的资源类型为应用和集群，微服务空间的相关权限操作需要在RAM配置。
- 配置管理ACM、应用实时监控服务ARMS等暂不支持资源组。

操作步骤

以上述场景示例为例，介绍如何通过资源管理和RAM联合进行权限控制。

1. 在资源管理控制台中创建资源组groupA和groupB。具体操作，请参见创建资源组。
2. 为资源组groupA和groupB添加集群和应用资源。具体操作，请参见资源转入到当前资源组。
3. 在RAM控制台中分别为部门deptA和deptB创建RAM用户subA和subB。具体操作，请参见创建RAM用户。
4. 在RAM控制台中创建通用的权限策略。具体操作，请参见创建自定义权限策略。

   策略内容如下：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
              "edas:*Cluster",
            "edas:*Application"
        ],
        "Resource": [
          "acs:edas:*:*:*"
        ]
      }
    ]
}
```

5. 在资源管理控制台中的资源组groupA和groupB中分别为RAM用户subA和subB授予通用权限策略。具体操作，请参见添加RAM身份并授权。
6. （可选）需要调整资源配置时，调整资源组即可。
7. （可选）为用户subA直接授予非资源组资源的权限，以微服务空间下微服务列表为例。具体操作，请参见为RAM用户授权。

   > **说明** 即使给资源组groupA绑定了AliyunEDASFullAccess并授予了subA，当subA使用微服务空间为条件查询微服务列表时，仍会出现无权限的提醒，因微服务空间不关联资源组，需要将微服务空间权限策略直接授予subA。

   策略内容如下：

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ReadService"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace"],
       "Effect": "Allow"
     }
   ]
 }
```

   同样，对于权限细分规则中的**管理集群**、**管理应用**和**微服务管理**（使用应用为条件） 之外的权限，也都需要直接授予subA相应的权限策略。

结果验证

在完成权限管理后，请根据您的实际业务需求验证RAM用户的权限是否符合预期。

7.2.4.5. 列表鉴权

列表鉴权功能默认开启，当RAM用户登录EDAS对相关资源进行List操作时，系统将进行RAM鉴权并只返回具有相关权限的资源访问结果。

背景信息

涉及微服务空间列表页、集群列表页、应用列表页的所有列表操作，例如：

- 创建应用时，选择微服务空间和集群时的相关列表操作。
- 创建集群时，选择微服务空间的相关列表操作。
- 查看微服务时，选择微服务空间的相关列表操作。

- 任务调度页面，选择微服务空间的相关列表操作。

列表鉴权用到的RAM Policy Action如下：

范围	权限
微服务空间	edas:ReadNamespace
集群	edas:ReadCluster
应用	edas:ReadApplication

> **说明** 只要拥有相关资源读权限的Action，即可在列表页看到该资源。

使用建议

推荐您使用**微服务空间-集群-应用**三者统一授权的方式进行资源管理。使用时有以下几点建议：

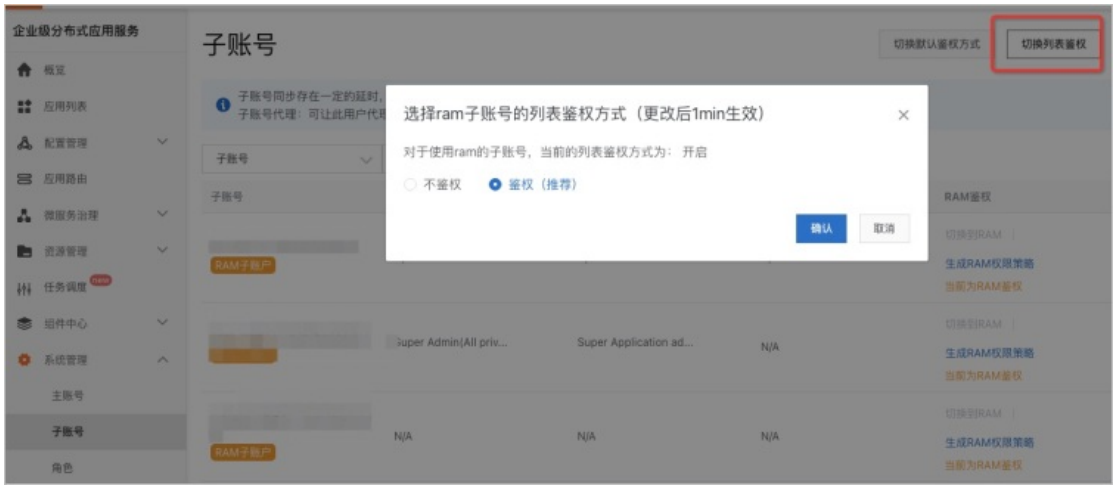
- 当您授权应用的读权限时，建议同时授予该应用所在集群、微服务空间的读权限。
- 当您授予集群的读权限时，建议同时授予该集群所在微服务空间的读权限。
- 考虑到RAM Policy有长度限制，建议您使用通配符授权，例如借助EDAS权限助手来配置某微服务空间下所有应用的操作权限。具体操作，请参见使用EDAS权限助手生成权限策略。
- 当您的资源数较多时，建议您使用资源组来管理您的资源。具体操作，请参见借助资源组进行权限管理。

> **说明** 资源组目前仅覆盖了应用和集群，微服务空间的相关权限依然需要在RAM控制台配置。

关闭列表鉴权

列表鉴权默认打开。可以通过**系统管理 > 子账号**页面的**切换列表鉴权**按钮进行关闭。

1. 登录EDAS控制台。
2. 在EDAS控制台的**系统管理 > 子账号**菜单内，在页面右上角单击**切换列表鉴权**。



3. 在弹出的对话框中单击**不鉴权**，再单击**确认**完成关闭。

> **说明** 关闭鉴权操作有延迟，切换后请等待1分钟，然后再次刷新界面可查看是否生效。

问题处理

当您遇到下述问题时，可根据指导进行处理。

- 问题一：如果某个应用AppX属于微服务空间nX，RAM用户subAccount拥有AppX的读（Read）权限，但是没有微

服务空间nX的Read权限，那在应用列表页能够找到该应用吗？
答：可以。在应用列表页，微服务空间选择所有微服务空间，即可看到该应用。

- 问题二：选择关闭了列表鉴权，为什么没有成功关闭？
  答：关闭操作大约有1分钟的延迟。当您关闭了页面之后，请1分钟后再检查是否成功关闭。
- 问题三：列表鉴权关闭后，怎么打不开某些资源的详情页了？
  答：列表鉴权关闭，并不影响原来的资源鉴权逻辑。列表鉴权关闭后，资源原来的鉴权逻辑依然存在，若子账号没有相关的权限，依然会被限制访问。

7.2.4.6. 使用标签控制资源访问

当EDAS的应用和集群绑定标签后，您可以使用标签控制资源的访问权限。本文以应用为例，介绍如何为RAM用户授予特定的策略，通过标签来控制RAM用户对应用的相关权限。

前提条件

- 在EDAS上已部署应用。具体操作，请参见：
  - K8s环境：创建和部署应用概述（K8s）
  - ECS环境：应用创建和部署概述（ECS）
- 在应用上绑定标签。具体操作，请参见使用标签查找资源。

背景信息

阿里云的用户权限是基于策略为管理主体的，您可以根据不同用户的职责配置RAM策略。在策略中，您可以定义多个标签，然后将一个或多个策略授权给RAM用户或用户组。如果要控制RAM用户可以访问哪些资源，您可以创建自定义策略并使用标签来实现访问控制。

使用标签控制资源的访问权限仅支持RAM鉴权，不支持EDAS内置的鉴权模式。更多权限管理信息，请参见权限管理概述。

假设在EDAS上已部署3个应用，这三个应用分别属于不同环境和不同项目，且所绑定的标签信息如下：

```
app-001:
    Enviroment=TEST  #测试环境
    Team=team1       #项目1
app-002:
    Enviroment=DEV   #开发环境
    Team=team1       #项目1
app-003:
    Enviroment=PROD  #生产环境
    Team=team2       #项目2
```

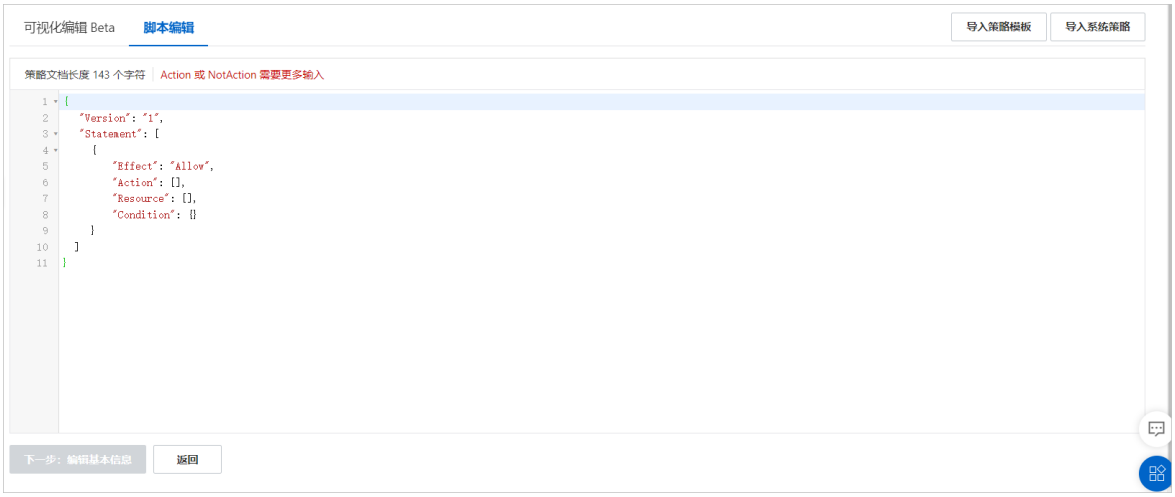
假设您有三个RAM用户（user1，user2和user3），为实现权限最小管控，您可以为RAM用户授予目标资源的访问权限。假设有以下三种场景：

- user1需要管理所有的开发环境和测试环境的应用。
- user2需要管理项目1下的所有测试环境的应用。
- user3需要访问除了生产环境以外所有的应用。

您可以使用标签来自定义权限策略，实现上述RAM用户的需求。

使用标签配置自定义权限策略

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏选择**权限管理 > 权限策略**。
3. 在**权限策略**页面，单击**创建权限策略**。
4. 在**创建权限策略**页面，单击**脚本配置**页签，然后在策略文档中自定义设置策略内容，单击**下一步**。



基于背景信息中列出的三个场景，本文提供的示例权限策略内容如下：

- user1需要管理所有的开发环境和测试环境的应用。

```
{
  "Statement": [
    {
      "Action": "edas:ManageApplication",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Enviroment": ["DEV", "TEST"]
        }
      }
    }
  ],
  "Version": "1"
}
```

- user2需要管理项目1下的所有测试环境的应用。

```
{
  "Statement": [
    {
      "Action": "edas:ManageApplication",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Team": ["team1"],
          "edas:tag/Enviroment": ["TEST"]
        }
      }
    }
  ],
  "Version": "1"
}
```

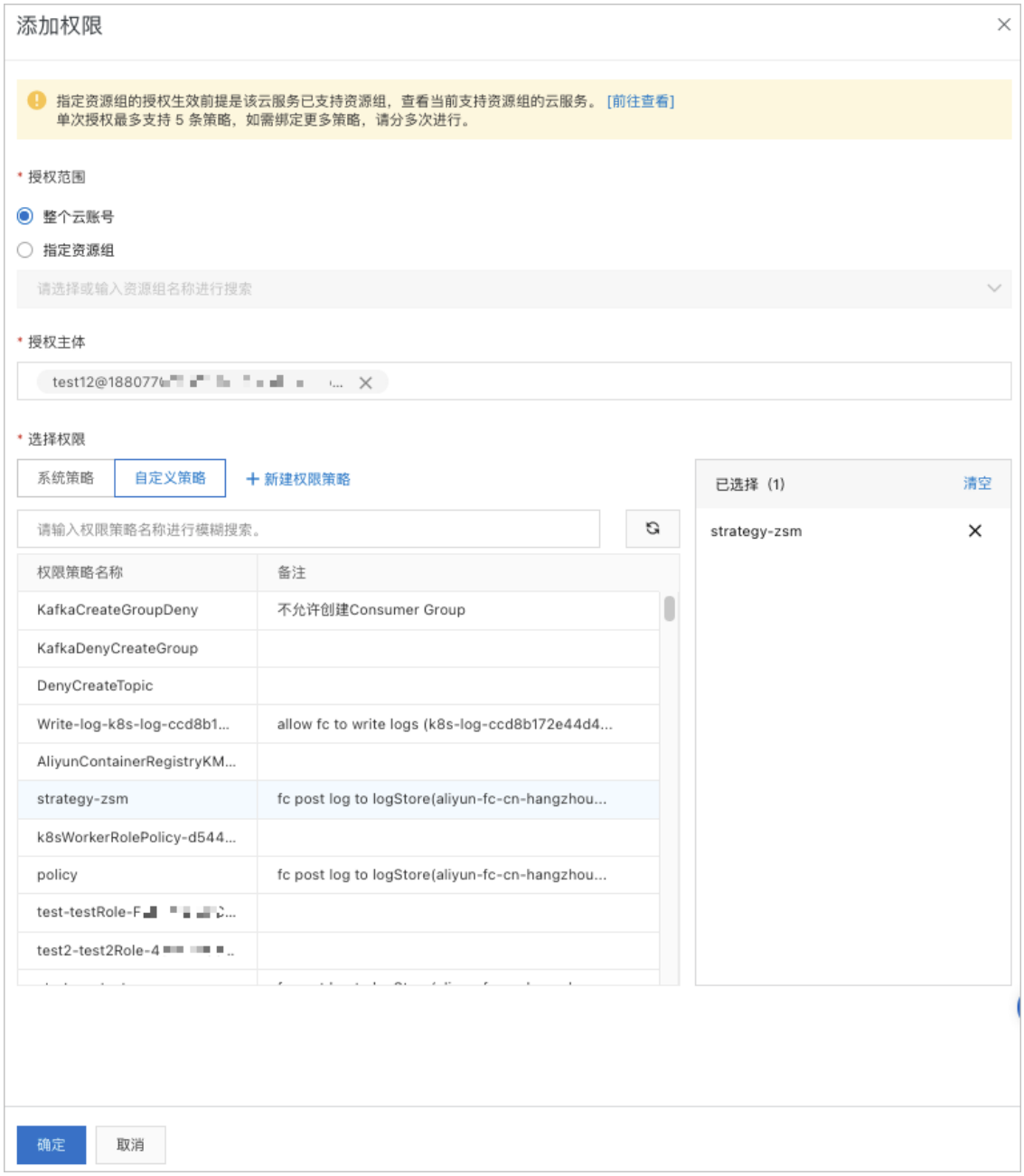
- user3需要访问除了生产环境以外所有的应用。

```
{
  "Statement": [
    {
      "Action": "edas:ReadApplication",
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": "edas:ReadApplication",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Enviroment": ["PROD"]
        }
      }
    }
  ],
  "Version": "1"
}
```

5. 输入策略**名称**和**（可选）备注**，然后在**策略内容**区域检查设置的策略内容，然后单击**确定**。
   添加完自定义权限策略后，新增策略出现在权限策略列表。

为RAM用户授权

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏选择**人员管理 > 用户**。
3. 在用户列表页面找到目标RAM用户，单击**操作**列下的**添加权限**。
4. 在**添加权限**面板的**授权范围**区域，单击**云账号全部资源**。
5. 在**添加权限**面板的**选择权限**区域，单击**自定义策略**，在搜索文本框搜索目标策略，然后单击目标策略，再单击**确定**。



6. 在**添加权限**面板，确认授权信息并单击**完成**。

RAM用户访问资源

RAM用户登录EDAS控制台，查看是否能访问目标资源。

7.2.4.7. 在RAM中配置服务测试相关权限

服务测试功能需要创建一个服务消费者，调用您的VPC中的服务提供者，从而测试服务提供者。本文介绍如何在RAM控制台对RAM用户授予这些操作的权限。

前提条件

服务测试采用RAM用户鉴权的模式，所以您需要先将EDAS内置授权切换为RAM授权。详情请参见将EDAS内置的权限管理切换为RAM权限管理。

创建测试服务的自定义权限策略并为RAM用户授权

RAM用户要测试服务，需要两个权限：edas:ReadService和edas:TestService。

1. 登录RAM控制台。
2. 在左侧导航栏选择**权限管理 > 权限策略**。
3. 在**权限策略**页面单击**创建权限策略**。
4. 在**创建权限策略**页面选择**脚本配置**，然后在策略文档区域输入测试服务的自定义权限策略，然后单击**下一步**。
   测试服务的自定义权限策略内容如下：

```
{
    "Statement": [
        {
            "Action": [
                "edas:ReadService"
            ],
            "Effect": "Allow",
            "Resource": [
                "acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"
            ]
        },
        {
            "Action": [
                "edas:TestService"
            ],
            "Effect": "Allow",
            "Resource": [
                "acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"
            ]
        }
    ],
    "Version": "1"
}
```

> **说明** `$namespace` 和 `$applicationId` 请替换为实际的微服务空间和应用。如果要测试所有微服务空间和应用的服务，将 `$namespace` 和 `$applicationId` 替换为星号（*）即可。

5. 填写基本信息后，单击**确定**。
   创建成功后，界面会提示 **自定义权限策略新建成功** 。
6. 为RAM用户授权创建的测试服务的自定义权限，详情请参见为RAM用户授权。

7.2.4.8. 如何自动为ECS实例添加访问ACM所需的RAM角色

借助ECS实例RAM角色，可实现无需配置AccessKey（AK）即可访问ACM，从而提高安全性。本文介绍了借助EDAS应用的挂载脚本能力，在扩容ECS实例到应用中时，自动为ECS实例添加访问ACM所需的RAM角色。

背景信息

以往，如果部署在ECS实例中的应用程序需要访问ACM，必须将AccessKey以配置文件或其他形式保存在ECS实例中，这在一定程度上增加了AccessKey管理的复杂性，并且降低了AccessKey的保密性。创建AccessKey的具体操作，请参见获取AccessKey。

现在，借助ECS实例RAM角色，您可以将RAM角色和ECS实例关联起来，然后将RAM角色名称告知ACM SDK（1.0.8及以上版本），此后无需配置AccessKey即可访问ACM。另外，借助RAM（访问控制），您可以通过角色和授权策略实现不同实例对ACM具有不同访问权限的目的。例如，如果配置只读策略，关联了该角色的ECS就只能读取ACM的配置，而无法新增或修改ACM配置。

前提条件

已开通访问控制（RAM），相关操作，请参见：计费方法。

步骤一：创建RAM角色并配置授权策略

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 角色**。
3. 在**角色**页面，单击目标RAM角色**操作**列的**添加权限**。
4. 在**添加权限**面板，为RAM角色添加权限。
   i. 选择授权应用范围。
      - **整个云账号**：权限在当前阿里云账号内生效。
      - **指定资源组**：权限在指定的资源组内生效。

      > **说明** 指定资源组授权生效的前提是该云服务已支持资源组。更多信息，请参见支持资源组的云服务。

   ii. 输入授权主体。

      授权主体即需要授权的RAM角色，系统会自动填入当前的RAM角色，您也可以添加其他RAM角色。

   iii. 选择权限策略。

      > **说明** 每次最多绑定5条策略，如需绑定更多策略，请分次操作。

5. 单击**确定**。
6. 单击**完成**。

> **说明** 您可以在**添加权限**对话框中，通过关键词搜索授权策略 `AliyunACMFullAccess` ，并单击该授权策略将其添加至右侧的**已选授**列表，然后单击**确定**添加权限。如果需要用到加解密配置功能，则还要添加 `AliyunKMSCryptoAdminAccess` 授权策略。

此时，该角色已具备ACM的所有操作权限。

步骤二：创建权限策略

1.
2. 单击**创建权限策略**。
3. 在**创建权限策略**页面选择**脚本配置**，然后在策略文档区域输入测试服务的自定义权限策略，然后单击**下一步**。

   测试服务的自定义权限策略内容如下：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ecs:AttachInstanceRamRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ecs:DescribeInstanceRamRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ram:*",
            "Resource": "acs:ram:*:<替换成您的主账号ID>:role/<替换成您创建的角色名称>"
        }
    ],
    "Version": "1"
}
```

4. 填写**策略名称**为AttachACMRamRoleToECSPolicy，并填写**备注**等基本信息。
   创建成功后，界面会提示 `自定义权限策略新建成功` 。
5.

步骤三：创建RAM用户并添加授权

1. 在左侧导航栏，选择**身份管理 > 用户**。
2. 在**用户**页面，单击**创建用户**。
3. 在**创建用户**页面的**用户账号信息**区域，输入**登录名称**和**显示名称**。

   > **说明** 单击**添加用户**，可一次性创建多个RAM用户。

4. 在**访问方式**区域下，选择**Open API 调用访问**，然后单击**确定**，RAM会自动为RAM用户创建AccessKey（API访问密钥），请记录下来供后续步骤使用并妥善保管。
5. 在**用户**页面，单击目标RAM用户**操作**列的**添加权限**。
6. 在**添加权限**面板设置**授权范围**。
7. 在**权限策略名称**上方的输入框内，输入步骤二：创建权限策略中生成的策略名称。
8. 单击**确定**。

步骤四：通过挂载脚本为ECS实例授予RAM角色

1. 登录EDAS控制台。
2. 在左侧导航栏中单击**应用列表**，在顶部菜单栏选择地域并在页面上方选择微服务空间，然后在**应用列表**页面单击具体的应用名称。
3. 在应用的**基本信息**页签的**应用设置**区域单击**编辑**，然后在下拉列表中单击**挂载脚本**。
4. 在**挂载脚本**对话框中单击展开**准备实例脚本**。
5. 在文本框中输入以下脚本，然后单击**修改**。

   > **注意** 以下脚本含有您在步骤三：创建RAM用户并添加授权中创建的子账号的访问密钥，请妥善保管。

```
#!/bin/sh
fileURL='https://edas-public.oss-cn-hangzhou.aliyuncs.com/samples/acm/attachAcmRamRole.sh'
file=/tmp/attachAcmRamRoleToEcs.sh
wget "$fileURL" -O "$file" &>/dev/null
chmod +x "$file"
#<accessKeyId>替换成子账号对应的AccessKey ID。
#<accessSecret>替换成子账号对应的AccessKey Secret。
#<ecsRamRoleForACM>替换成您所创建的角色名称。
bash "$file" <accessKeyId> <accessSecret> <ecsRamRoleForACM>
```

该脚本执行后会将<ecsRamRoleForACM>绑定到扩容的ECS实例上。

更多信息

- 获取AccessKey
- RAM（访问控制）
- 创建可信实体为阿里云服务的RAM角色
- 使用实例RAM角色访问其他云产品
- ACM Java Native SDK概述

7.2.4.9. 权限策略示例库

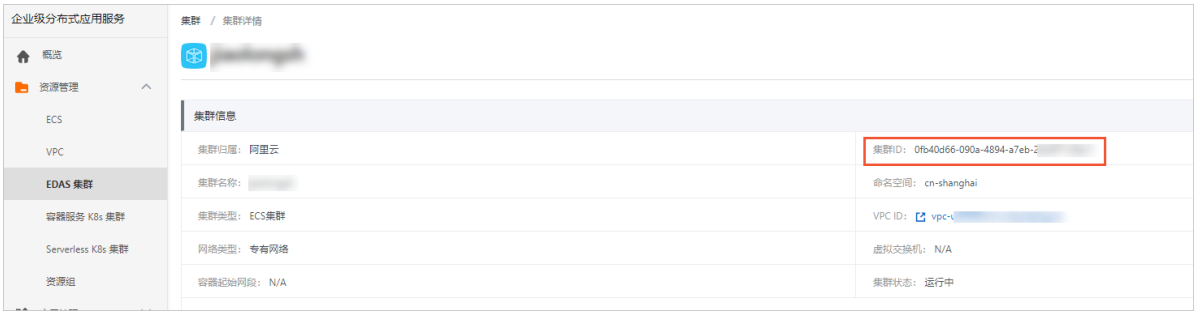
文本介绍了在RAM中可以授权的跟EDAS相关的权限细分规则和权限策略。

权限策略中Resource的变量说明

权限策略声明中Resource包含的变量说明如下：

- $regionid：所在地域的ID，如cn-shanghai，具体取值，参见地域和可用区。
- $namespace：微服务空间租户ID，详情可在EDAS控制台左侧导航栏选择**资源管理 > 微服务空间**，进入微服务空间页面获取，以下截图圈住部分即为微服务空间的租户ID。

- $clusterId：集群ID，如8c349f69-505c-436f-8dc7-**********，您可在集群详情页面获取。



- $applicationId：应用ID，如ec8e38a3-3dca-47a7-b6f9-5**********，具体可在应用基本信息页面获取。

权限细分规则

本章节介绍了EDAS内置权限和RAM中权限策略的对应关系。

管理微服务空间

Code	Description	Dependency action	Resource
1.1	创建微服务空间	edas:CreateNamespace	acs:edas:$regionid:$accountid:namespace/*
1.2	删除微服务空间	edas:ReadNamespace edas:DeleteNamespace	acs:edas:$regionid:$accountid:namespace/$namespace
1.4	编辑微服务空间	edas:ManageNamespace edas:ReadNamespace	acs:edas:$regionid:$accountid:namespace/$namespace

管理集群

Code	Description	Dependency action	Resource
2.1	创建集群	edas:CreateCluster	acs:edas:$regionid:$accountid:namespace/$namespace/cluster/*
2.2	删除集群	edas:ReadCluster edas:DeleteCluster	acs:edas:$regionid:$accountid:namespace/$namespace/cluster/$clusterId
2.4	管理集群	edas:ReadCluster edas:ManageCluster	acs:edas:$regionid:$accountid:namespace/$namespace/cluster/$clusterId
2.3	查看集群	edas:ReadCluster	acs:edas:$regionid:$accountid:namespace/$namespace/cluster/$clusterId

管理应用

Code	Description	Dependency action	Resource
3.1	创建应用	edas:CreateApplication	acs:edas:$regionid:$accountid:namespace/$namespace/application/*

Code	Description	Dependency action	Resource
3.2	删除应用	edas:ReadApplication edas:DeleteApplication	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
3.3	查看应用	edas:ReadApplication	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
3.4	管理应用	edas:ManageApplication edas:ReadApplication	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
3.5	配置应用	edas:ConfigApplication edas:ReadApplication	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
3.6	管理日志	edas:ReadApplication edas:ManageAppLog	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId

微服务管理

Code	Description	Dependency action	Resource
4.1	查看服务	edas:ReadService	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
4.2	测试服务	edas:TestService	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId
4.3	管理服务	edas:ReadService edas:ManageService	acs:edas:$regionid:$accountid:namespace/$namespace/application/$applicationId

配置管理

Code	Description	Dependency action	Resource
5.1	查看配置	acms:R	acs:acms:$regionid:$accountid:cfg/$namespace/$groupId/$configId
5.2	管理配置	acms:*	acs:acms:$regionid:$accountid:cfg/$namespace/$groupId/$configId

系统管理

Code	Description	Dependency action	Resource
6.1	EDAS系统管理	edas:ManageSystem	acs:edas:$regionid:$accountid:*
6.2	查看操作日志	edas:ReadOperationLog	acs:edas:$regionid:$accountid:*
6.3	系统运维	edas:ManageOperation	acs:edas:$regionid:$accountid:*
6.4	ECS代购	edas:ECSPurchase	acs:edas:*:*:*
6.5	SLB代购	edas:SLBPurchase	acs:edas:*:*:*
6.6	SLS代购	edas:SLSPurchase	acs:edas:*:*:*

EDAS商用相关功能

Code	Description	Dependency action	Resource
7	EDAS商用相关功能	edas:ManageCommercialization	acs:edas:$regionid:$accountid:*

管理微服务空间

集群授权

跟集群使用相关的场景授权说明如下：

集群相关的只读权限，例如查看单个集群详情，查看集群下的实例或应用等信息。

> **说明** 通过给RAM用户授予资源组，RAM用户才能在集群列表中查看到被授权的集群。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ReadCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"],
       "Effect": "Allow"
      }
   ]
 }
```

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ReadCluster","edas:DeleteCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"],
       "Effect": "Allow"
      }
   ]
 }
```

注意 创建集群时，`Resource` 中 `cluster/` 后面的必须是星号（*）。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:CreateCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/*"],
       "Effect": "Allow"
      }
   ]
 }
```

包括创建集群、导入实例到集群、编辑集群信息和删除集群等。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"],
       "Effect": "Allow"
      }
   ]
 }
```

下面以两个示例场景来说明如何配置集群管理的权限策略。

- **场景一**：授权集群的管理权限，但不允许创建集群。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"],
       "Effect": "Allow"
     },
     {
       "Action": ["edas:CreateCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/*"],
       "Effect": "Deny"
      }
   ]
 }
```

说明 其中$clusterId若配置为具体的值，则只能管理对应ID的集群；若配置为星号（*），则能管理指定微服务空间下的所有集群。

- **场景二**：授予集群的管理权限，但是不允许创建和删除集群。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"],
       "Effect": "Allow"
     },
     {
       "Action": ["edas:CreateCluster","edas:DeleteCluster"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace/cluster/*"],
       "Effect": "Deny"
      }
   ]
 }
```

查看集群详情

删除集群

创建集群

管理集群

微服务空间

跟微服务空间使用相关的场景授权说明如下：

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ReadNamespace","edas:DeleteNamespace"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace"],
       "Effect": "Allow"
      }
   ]
 }
```

> 注意　创建微服务空间时，`Resource` 中 `namespace/` 后面的必须是星号（*）。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:CreateNamespace"],
       "Resource": ["acs:edas:$regionid:*:namespace/*"],
       "Effect": "Allow"
      }
   ]
 }
```

当编辑或修改微服务空间名称时，需授予管理微服务空间权限。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageNamespace"],
       "Resource": ["acs:edas:$regionid:*:namespace/$namespace"],
       "Effect": "Allow"
      }
   ]
 }
```

删除微服务空间

创建微服务空间

管理微服务空间

资源代购

为增强企业级用户的权限管理能力，EDAS支持资源代购的RAM管理权限。主要涉及的资源包含ECS、SLB和SLS。

资源代购的场景授权说明如下：

> 注意
>
> - 所有的代购类权限策略配置中，`resource` 必须是 `"acs:edas:*:*:*"` ，暂不支持更细粒度配置。
> - 资源代购权限策略只适用于RAM用户。

- 使用范围：
  - 在ECS集群内代购ECS
  - 在ECS集群内创建应用时代购ECS
  - 在ECS集群内扩容应用时代购ECS
- 策略示例：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "edas:ECSPurchase"
      ],
      "Resource": [
        "acs:edas:*:*:*"
      ]
    }
  ]
}
```

- 使用范围：为应用绑定负载均衡时代购SLB。
- 策略示例：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "edas:SLBPurchase"
      ],
      "Resource": [
        "acs:edas:*:*:*"
      ]
    }
  ]
}
```

- 使用范围：为应用代购日志服务SLS。
- 策略示例：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "edas:SLSPurchase"
      ],
      "Resource": [
        "acs:edas:*:*:*"
      ]
    }
  ]
}
```

ECS代购的权限

SLB代购的权限

SLS代购的权限

应用管理

跟应用相关的场景授权说明如下：

- **管理应用**：使用RAM用户可以查看该应用的基本信息、管理配置和应用日志，但不允许创建和删除应用。

```
{
    "Statement": [
      {
        "Action": [
          "edas:*Application"
        ],
        "Effect": "Allow",
        "Resource": ["acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"]
      },
      {
        "Action": [
          "edas:DeleteApplication"
        ],
        "Resource":["acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"],
        "Effect": "Deny"
      },
      {
        "Action": [
          "edas:CreateApplication"
        ],
        "Resource":["acs:edas:$regionid:*:namespace/$namespace/application/*"],
        "Effect": "Deny"
      }
    ],
    "Version": "1"
}
```

- **创建应用**

> **注意** 创建应用时使用的是所在集群内的实例，因此还需要授予查看集群的权限。

```
{
  "Statement": [
    {
      "Action": [
        "edas:CreateApplication",
        "edas:ReadCluster"
      ],
      "Effect": "Allow",
      "Resource": [
              "acs:edas:$regionid:*:namespace/$namespace/application/*",
          "acs:edas:$regionid:*:namespace/$namespace/cluster/$clusterId"
      ]
    }
  ],
  "Version": "1"
}
```

- **删除应用**

> **注意** 删除应用时还需要授予查看应用的权限，这样才能找到需要删除的应用。

```
{
  "Statement": [
    {
      "Action": [
        "edas:DeleteApplication",
        "edas:ReadApplication"
      ],
      "Effect": "Allow",
      "Resource": ["acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"]
    }
  ],
  "Version": "1"
}
```

- **日志管理**

  > **注意** 管理应用日志时还需要授予查看应用的权限，这样才能找到需要日志管理的应用。

```
{
  "Statement": [
    {
      "Action": [
        "edas:ReadApplication",
        "edas:ManageAppLog"
      ],
      "Effect": "Allow",
      "Resource": ["acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"]
    }
  ],
  "Version": "1"
}
```

- **设置应用**：应用的设置包括对应用端口、Tomcat context、负载均衡、健康检查、JVM参数和同可用区优先等功能进行设置。

  > **注意** 设置应用时还需要授予查看应用的权限。

```
{
  "Statement": [
    {
      "Action": [
        "edas:ReadApplication",
        "edas:ConfigApplication"
      ],
      "Effect": "Allow",
      "Resource": ["acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"]
    },
  ],
  "Version": "1"
}
```

- **查看应用**：授权查看某个地域内的应用列表。

  > **说明** 一个地域下包括多个微服务空间，查看应用可以查看该某地域的所有微服务空间内的应用列表。

```
{
    "Statement": [
      {
        "Action": [
          "edas:ReadApplication"
        ],
        "Effect": "Allow",
        "Resource": ["acs:edas:$regionid:*:namespace/*/application/*"]
      }
    ],
    "Version": "1"
}
```

- **查看应用**：授权查看某命令空间内的应用列表。

```
{
    "Statement": [
      {
        "Action": [
          "edas:*Application",
          "edas:ReadCluster"
        ],
        "Effect": "Allow",
        "Resource": [
            "acs:edas:$regionid:*:namespace/$namespace/application/*",
          "acs:edas:$regionid:*:namespace/$namespace/cluster/*"
         ]
      }
    ],
    "Version": "1"
}
```

单个应用的授权

多个应用的授权

微服务管理

与微服务管理使用相关的场景授权说明如下：

> **说明** 如果要查询所有应用服务，您可将下述权限策略中的 `$applicationId` 替换为星号（*）。

```
{
    "Statement": [
      {
        "Action": [
          "edas:ReadService"
        ],
        "Effect": "Allow",
        "Resource": [
            "acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"
            ]
      }
    ],
    "Version": "1"
}
```

说明 如果要测试所有微服务空间和应用服务，您可将下述权限策略中的 `$namespace` 和 `$applicationId` 替换为星号（*）。

```
{
    "Statement": [
      {
        "Action": [
          "edas:TestService"
        ],
        "Effect": "Allow",
        "Resource": [
            "acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"
            ]
      }
    ],
    "Version": "1"
}
```

说明 如果要授权所有应用，您可将下述权限策略中的 `$applicationId` 替换为星号（*）。

```
{
    "Statement": [
      {
        "Action": [
          "edas:ManageService"
        ],
        "Effect": "Allow",
        "Resource": [
            "acs:edas:$regionid:*:namespace/$namespace/application/$applicationId"
        ]
      }
    ],
    "Version": "1"
}
```

注意 离群摘除影响到微服务空间下的相关应用，所以授权粒度为微服务空间。

```
{
    "Statement": [
      {
        "Action": [
          "edas:ManageService"
        ],
        "Effect": "Allow",
        "Resource": [
            "acs:edas:$regionid:*:namespace/$namespace"
        ]
      }
    ],
    "Version": "1"
}
```

查询服务

测试服务

服务鉴权

离群实例摘除

配置管理

EDAS集成了ACM的配置管理能力，与配置管理相关的权限控制，请参见访问权限控制。

系统管理

系统管理包含了管理RAM用户、查看产品用量、查看操作日志等系统级的权限。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageSystem"],
       "Resource": ["acs:edas:*:*:*"],
       "Effect": "Allow"
      }
   ]
 }
```

> **说明** 系统权限不涉及具体的资源，故权限策略中的 `Resource` 直接用 `acs:edas:*:*:*` 即可。

系统运维包含了查看操作日志、执行批量运维和管理资源组的权限。

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ManageOperation"],
       "Resource": ["acs:edas:*:*:*"],
       "Effect": "Allow"
      }
   ]
 }
```

```
{
   "Version": "1",
   "Statement": [
     {
       "Action": ["edas:ReadOperationLog"],
       "Resource": ["acs:edas:*:*:*"],
       "Effect": "Allow"
      }
   ]
 }
```

系统运维

操作日志

7.2.5. 管理EDAS内置权限（不推荐）

为了能够统一管理EDAS和其他阿里云产品的权限，EDAS已经支持将内置权限管理迁移到RAM。在未完成迁移前，您可以继续使用EDAS内置的权限管理操作。

背景信息

EDAS目前处于EDAS内置权限管理与RAM权限管理共存的过渡状态，目前您的RAM用户或子账号所使用的权限管理规则如下：

- 未使用过EDAS内置权限管理的子账号，直接使用RAM权限管理，不能再使用EDAS内置权限管理。
- 使用过EDAS内置权限管理的子账号：
  - 拥有RAM中AliyunEDASFullAccess权限的子账号，直接使用RAM鉴权，不再开放EDAS内置鉴权。
  - 使用过EDAS内置权限管理的子账号，建议手动切换为RAM鉴权，具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。如果不做任何改动，可继续使用EDAS内置鉴权。

创建角色

云账号可以通过创建不同的角色，定义不同的操作权限。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 角色**。
3. 在**角色**页面右上角单击**创建角色**。
4. 在**创建角色**页面输入**角色名称**，将左侧**可选权限**列表中选择权限，单击**添加 >>**，添加到右侧的**已选权限**列表，然后单击**确定**。

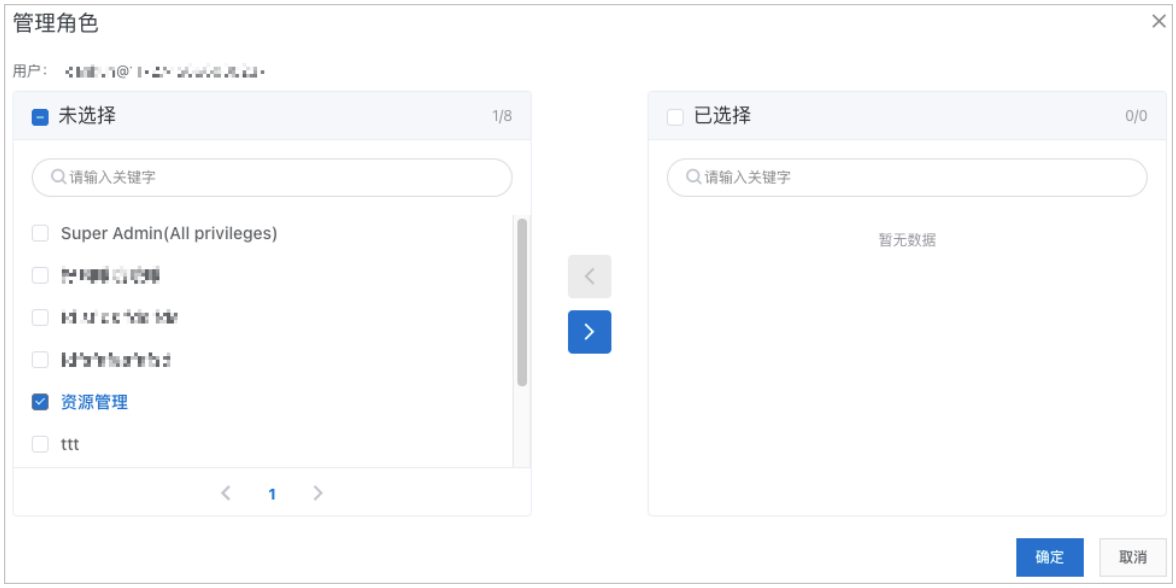
返回**角色**页面，创建的角色会显示在角色列表中。
您可以在目标角色的**操作**列单击**查看权限**、**管理权限**或**删除**，对角色进行相关的操作。

为子账号授予角色

一般需要为子账号授予相应角色，才能使子账号对该应用有操作权限。

1. 登录EDAS控制台。

2. 在左侧导航栏选择**系统管理 > 子账号**。
3. 在目标子账号的**操作**列单击**管理角色**。
4. 在**管理角色**对话框左侧的**未选择**列表中选择角色，单击>，添加到右侧的**已选择**列表中，然后单击**确定**。

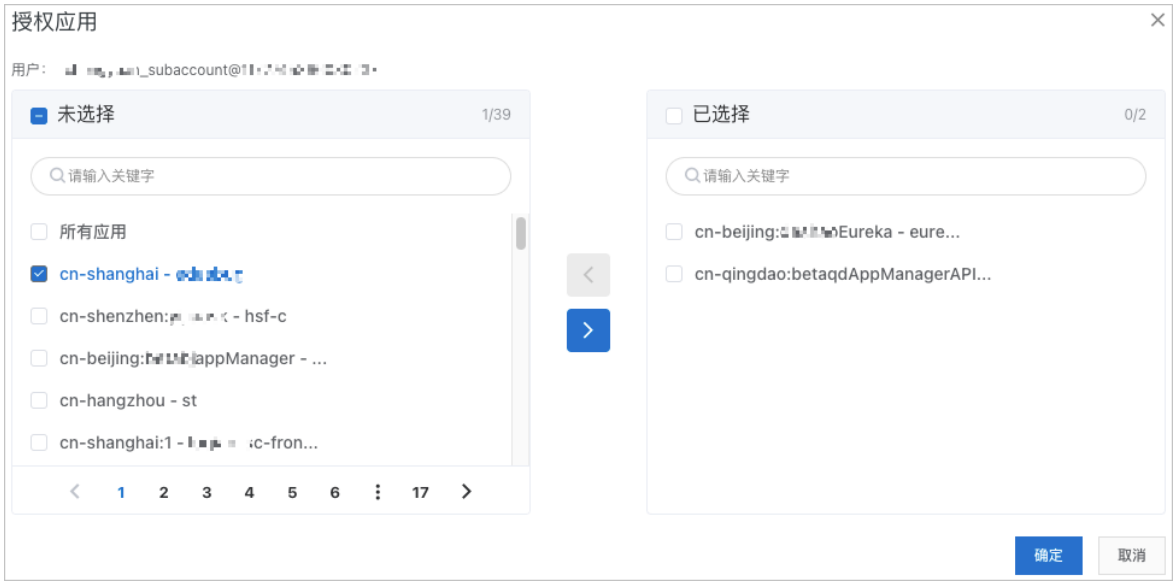


返回**子账号**页面，授权的角色会显示在该子账号的**角色**列中。
如需调整授权的角色，可以重复授权应用的操作步骤，在**管理角色**对话框中调整两侧列表中的角色。

为子账号授权应用

授权应用后，只表明子账号对应用具有权限，至于对该应用的具体操作，如应用启动、删除应用等，需要通过授权角色来授予操作的权限。所以授权应用后，一般需要再为子账号授予相应角色，才能使子账号对该应用有操作权限。

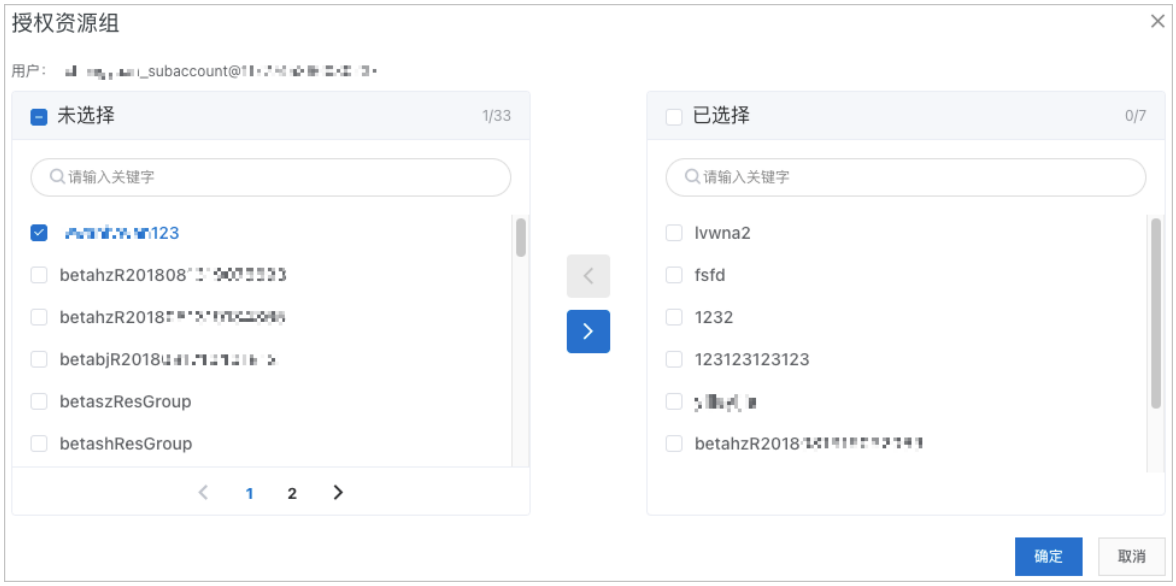
1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 子账号**。
3. 在目标子账号的**操作**列单击**授权应用**。
4. 在**授权应用**对话框左侧的**未选择**列表中选择应用，单击>，添加到右侧的**已选择**列表中，然后单击**确定**。



返回**子账号**页面，授权的应用会显示在该子账号的**授权应用**列中。
如需调整授权的应用，可以重复授权应用的操作步骤，在**授权应用**对话框中调整两侧列表中的应用。

为子账号授权资源组

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 子账号**。
3. 在目标子账号的**操作**列单击**授权资源组**。
4. 在**授权资源组**对话框左侧的**未选择**列表中选择资源组，单击>，添加到右侧的**已选择**列表中，然后单击**确定**。



返回**子账号**页面，授权的资源组会显示在该子账号的**授权资源组**列中。
如需调整授权的资源组，可以重复授权应用的操作步骤，在**授权资源组**对话框中调整两侧列表中的资源组。

7.2.6. 使用EDAS权限助手生成权限策略

EDAS权限助手是一个RAM权限策略的生成工具，用于帮助您在RAM中快速创建EDAS相关的权限策略，以便尽快将EDAS内置的子账号的权限管理迁移到RAM用户的权限管理。

查看EDAS系统权限策略模板

在EDAS控制台的权限助手页面，内置了8种默认的权限策略模板。您可以根据要迁移的内置子账号的功能，复制对应的系统权限策略，然后登录RAM控制台授权给对应的RAM用户，详情请参见将EDAS内置的权限管理切换为RAM权限管理。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 权限助手**。
3. 在**权限助手**页面中查看EDAS提供的系统权限策略模板。

   策略类型为**系统策略**，表示该权限策略为EDAS提供的权限策略模板，您可以根据实际需求执行以下操作：

   - 在策略最右侧单击**克隆**，进入**创建权限策略**面板，根据您的实际需求编辑、创建一个自定义权限策略。
   - 在策略最右侧单击**查看详情**，进入**查看详情**对话框，单击**复制**，然后登录RAM控制台，在创建自定义权限策略时使用，并授权给对应的RAM用户。具体操作，请参见将EDAS内置的权限管理切换为RAM权限管理。

EDAS自带的8种系统策略模板的详细介绍请参见EDAS系统策略模板介绍。

创建自定义权限策略模板

除了自带的系统策略模板外，你还可以通过权限助手创建自定义策略模板。下面以一个具体的示例，来介绍如何生成自定义的权限策略模板。

需要为某个RAM用户配置以下权限：

- 在华北2（北京）地域下，查看微服务空间test的权限。
- 在华北2（北京）地域下，查看微服务空间test下所有集群的权限。

- 在微服务空间test下，除创建应用外的其它所有应用相关的操作权限。
  1. 登录EDAS控制台。
  2. 在左侧导航栏选择**系统管理 > 权限助手**。
  3. 在**权限助手**页面单击**创建权限策略**。
  4. 在**创建权限策略**配置向导的**创建自定义权限策略**页签中设置权限策略的**策略名称**和**备注**。
  5. 添加权限语句。

     > **注意**
     > - 在新增权限语句时，只能选择一种类型（允许或拒绝）的权限效力。
     > - 您可以为一个权限策略创建多个权限语句，当某个权限的权限效力在不同权限语句中被设置为**允许**和**拒绝**时，遵循拒绝优先原则。

     i. 在**创建自定义权限策略**页签中单击**新增权限语句**，在**加授权语句**面板中，将查看微服务空间test、查看微服务空间test下的所有集群，以及操作微服务空间test下所有应用的**权限效力**设置为**允许**，然后单击**确认**。
        a. 在**权限效力**下方选择**允许**。
        b. 在**操作与资源授权**左侧的权限列表中选择**命名空间 > 查看命名空间**，在右侧的资源列表中选择*华北2（北京）*和*test*。

        c. 在**操作与资源授权**左侧的权限列表中选择**集群 > 查看集群**，在右侧的资源列表中选择*华北2（北京）*、*test*和*全部集群*。

d. 在**操作与资源授权**左侧的权限列表中选择**应用**（该操作会选中应用下的所有权限），在右侧的资源列表中选择*华北2（北京）*和*test*。

ii. 在**创建自定义权限策略**页签中单击**新增权限语句**，在**加授权语句**面板中，将微服务空间test下创建应用的**权限效力**设置为**拒绝**，然后单击**确认**。

a. 在**权限效力**下方选择**拒绝**。

b. 在**操作与资源授权**左侧的权限列表中选择**应用 > 创建应用**，在右侧的资源列表中选择*华北2（北京）*和*test*。

6. 在**策略预览**页签预览权限，然后单击**完成**。
控制台面板将会提示 新增策略授权成功 ，单击**返回列表查看**可返回**权限助手**页面，查看新建的权限策略模板。
在策略最右侧单击**查看详情**，进入**查看详情**对话框，单击**复制**，然后登录RAM控制台，在创建自定义权限策略时使用，并授权给对应的RAM用户。具体操作，情请参见将EDAS内置的权限管理切换为RAM权限管理。

EDAS系统策略模板介绍

超级管理员拥有EDAS的所有权限，其权限范围等同于主账号，在非必要时不推荐使用，请授予子账号更细粒度的权限策略。超级管理员拥有的权限如下：

- 微服务空间相关的所有权限
- 集群相关的所有权限
- 应用相关的所有权限
- 微服务相关的所有权限
- 管理配置相关的所有权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Namespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Cluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Application"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Service"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
              "edas:ManageSystem",
            "edas:ManageOperation",
            "edas:ReadOperationLog"
        ],
        "Resource": [
          "acs:edas:*:*:*"
        ]
      }
    ]
}
```

应用管理员拥有对应用操作的所有权限，拥有的权限如下：

- 应用相关的所有权限
- 微服务相关的所有权限

- 集群的所有权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Application"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Service"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
          "edas:ReadCluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      }
    ]
}
```

应用运维员通常负责应用的运维，相比于应用管理员，应用运维员无法创建或删除应用，只能对现有的应用进行运维。拥有的权限如下：

- 除了应用创建外的所有应用相关权限
- 微服务相关的所有权限
- 系统运维权限（edas:ManageOperation）

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Application"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Service"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:ManageOperation"
        ],
        "Resource": [
          "acs:edas:*:*:*"
        ]
      },
        {
        "Effect": "Deny",
        "Action": [
          "edas:CreateApplication"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      }
    ]
}
```

应用查看人员可以查看所有的应用信息。拥有的权限如下：

- 应用的查看权限
- 微服务的查看权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadApplication"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
          "edas:ReadService"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      }
    ]
}
```

资源管理员拥有微服务空间和集群的所有权限，它不负责应用的创建维护等，它只关心在EDAS上的资源管理。比如微服务空间的维护、管理ECS集群内的ECS资源等。拥有的权限如下：

- 微服务空间相关的所有权限
- 集群相关的所有权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Namespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
          "edas:*Cluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      }
    ]
}
```

相比于资源管理员，资源运维人员无法创建微服务空间和集群，仅能对当前的资源进行管理。拥有的权限如下：

- 除微服务空间创建外的所有微服务空间相关权限
- 除创建集群外的所有集群相关权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Namespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:*Cluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      },
      {
        "Effect": "Deny",
        "Action": [
          "edas:CreateNamespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
        {
        "Effect": "Deny",
        "Action": [
          "edas:CreateCluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      }
    ]
}
```

资源查看人员只可以对微服务空间和集群进行查看。拥有的权限如下：

- 微服务空间的查看权限
- 集群的查看权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadNamespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
          "edas:ReadCluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      }
    ]
}
```

EDAS只读权限拥有EDAS上所有资源的查看权限，拥有的权限如下：

- 微服务空间的读权限
- 集群的读权限
- 应用的读权限
- 微服务的读权限
- 配置的读权限
- 操作日志的读权限

其等效的RAM权限策略为：

```
{
    "Version": "1",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadNamespace"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadCluster"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/cluster/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadApplication"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "edas:ReadService"
        ],
        "Resource": [
          "acs:edas:*:*:namespace/*/application/*"
        ]
      },
        {
        "Effect": "Allow",
        "Action": [
          "edas:ReadOperationLog"
        ],
        "Resource": [
          "acs:edas:*:*:*"
        ]
      }
    ]
}
```

超级管理员

应用管理员

应用运维人员

应用查看人员

资源管理人员

资源运维人员

资源查看人员

EDAS只读权限

相关文档

- 权限管理概述
- 权限策略示例库
- 将EDAS内置的权限管理切换为RAM权限管理

7.3. 标签管理

7.3.1. 标签概述

标签是云资源的标识，可以帮助您从不同维度对具有相同特征的云资源进行分类，让资源管理变得更加轻松。

应用场景

标签的使用场景主要有：

- 使用标签查找资源
  您可以通过标签对应用和集群于进行标记和分类，通过控制台快速查找指定标签的资源。更多信息，请参见使用标签查找资源。
- 使用标签控制资源的访问
  标签与访问控制（RAM）的结合使用，能够让不同的RAM用户拥有不同云资源的访问和操作权限。更多信息，请参见使用标签控制资源访问。

使用限制

- 单个云资源最多可以绑定20个标签。
- 不同地域中的标签信息不互通。例如：在华东 1（杭州）地域创建的标签在华东 2（上海）地域不可见。

7.3.2. 使用标签查找资源

EDAS支持标签管理功能。您可以通过标签对应用和集群于进行标记和分类，便于资源的搜索和聚合。本文以容器服务K8s集群绑定标签为例，介绍如何绑定标签和使用标签查找资源。

背景信息

标签是云资源的标识，可以帮助您从不同的维度对具有相同特征的云资源进行分类、搜索和聚合，使资源管理更加容易。每个云资源均支持绑定多个标签。

随着您部署的应用、导入的K8s集群和创建的ECS集群的增多，您会发现利用标签将资源进行分组管理和归类有利于搜索和批量操作。

- 您可以根据不同的环境绑定不同的标签。
  - 测试环境：绑定一个类似Enviroment=TEST的标签键值对。
  - 开发环境：绑定一个类似Enviroment=DEV的标签键值对。
  - 生产环境：绑定一个类似Enviroment=PROD的标签键值对。
- 您可以根据不同的团队或者项目绑定不同的标签。
  - 一号团队或项目：绑定一个类似Team=team1的标签键值对。
  - 二号团队或项目：绑定一个类似Team=team2的标签键值对。

绑定标签

1. 登录EDAS控制台。
2. 在左侧导航栏选择**资源管理 > 容器服务K8s集群**。
3. 在**容器服务K8s集群**页面的顶部菜单栏选择目标地域，在页面中选择目标微服务空间。
4. 在集群列表中找到目标集群，单击标签列下的图标，然后单击**编辑标签**。
5. 在**编辑标签**对话框，单击**新增**，然后设置**标签键**和**标签值**，然后单击**确定**。

> **说明** 标签由一个键值对（Key-Value）组成，包含标签键（Key）、标签值（Value）。添加标签时，您需要了解以下几点：
>
> - **标签键**不能以*acs:*、*http://*和*https://*开头。
> - **标签键**和**标签值**最多支持128个字符，支持英文字母、数字和短划线（-）、半角逗号（,）、星号（*）、正斜线（/）、半角问号（?）和半角冒号（:）的组合。
> - 同一个资源中，不得出现相同的**标签键**。
> - 每个集群最多支持添加20个标签。

查找资源

1. 在**容器服务K8s集群**页面，单击页面中央的**标签**。
2. 在弹出的对话框，选择**标签键**和**标签值**，单击**搜索**。
   目标资源被筛选出来。

7.3.3. 使用标签控制资源访问

当EDAS的应用和集群绑定标签后，您可以使用标签控制资源的访问权限。本文以应用为例，介绍如何为RAM用户授予特定的策略，通过标签来控制RAM用户对应用的相关权限。

前提条件

- 在EDAS上已部署应用。具体操作，请参见：
  - K8s环境：创建和部署应用概述（K8s）
  - ECS环境：应用创建和部署概述（ECS）

- 在应用上绑定标签。具体操作，请参见使用标签查找资源。

背景信息

阿里云的用户权限是基于策略为管理主体的，您可以根据不同用户的职责配置RAM策略。在策略中，您可以定义多个标签，然后将一个或多个策略授权给RAM用户或用户组。如果要控制RAM用户可以访问哪些资源，您可以创建自定义策略并使用标签来实现访问控制。

使用标签控制资源的访问权限仅支持RAM鉴权，不支持EDAS内置的鉴权模式。更多权限管理信息，请参见权限管理概述。

假设在EDAS上已部署3个应用，这三个应用分别属于不同环境和不同项目，且所绑定的标签信息如下：

```
app-001:
    Enviroment=TEST   #测试环境
    Team=team1        #项目1
app-002:
    Enviroment=DEV    #开发环境
    Team=team1        #项目1
app-003:
    Enviroment=PROD   #生产环境
    Team=team2        #项目2
```

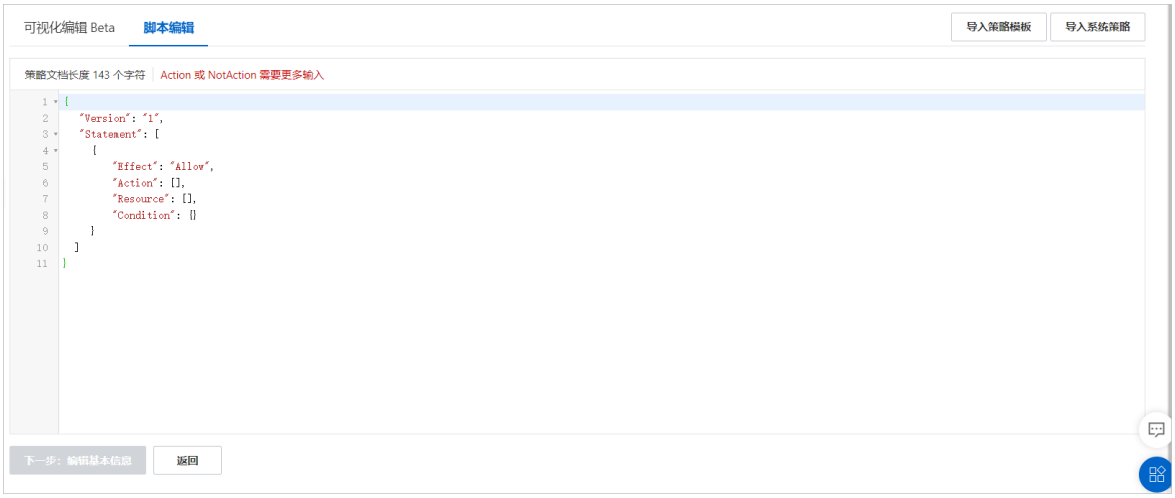
假设您有三个RAM用户（user1，user2和user3），为实现权限最小管控，您可以为RAM用户授予目标资源的访问权限。假设有以下三种场景：

- user1需要管理所有的开发环境和测试环境的应用。
- user2需要管理项目1下的所有测试环境的应用。
- user3需要访问除了生产环境以外所有的应用。

您可以使用标签来自定义权限策略，实现上述RAM用户的需求。

使用标签配置自定义权限策略

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏选择**权限管理 > 权限策略**。
3. 在**权限策略**页面，单击**创建权限策略**。
4. 在**创建权限策略**页面，单击**脚本配置**页签，然后在策略文档中自定义设置策略内容，单击**下一步**。



基于背景信息中列出的三个场景，本文提供的示例权限策略内容如下：

- user1需要管理所有的开发环境和测试环境的应用。

```
{
  "Statement": [
    {
      "Action": "edas:ManageApplication",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Enviroment": ["DEV", "TEST"]
        }
      }
    }
  ],
  "Version": "1"
}
```

- user2需要管理项目1下的所有测试环境的应用。

```
{
  "Statement": [
    {
      "Action": "edas:ManageApplication",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Team": ["team1"],
          "edas:tag/Enviroment": ["TEST"]
        }
      }
    }
  ],
  "Version": "1"
}
```

- user3需要访问除了生产环境以外所有的应用。

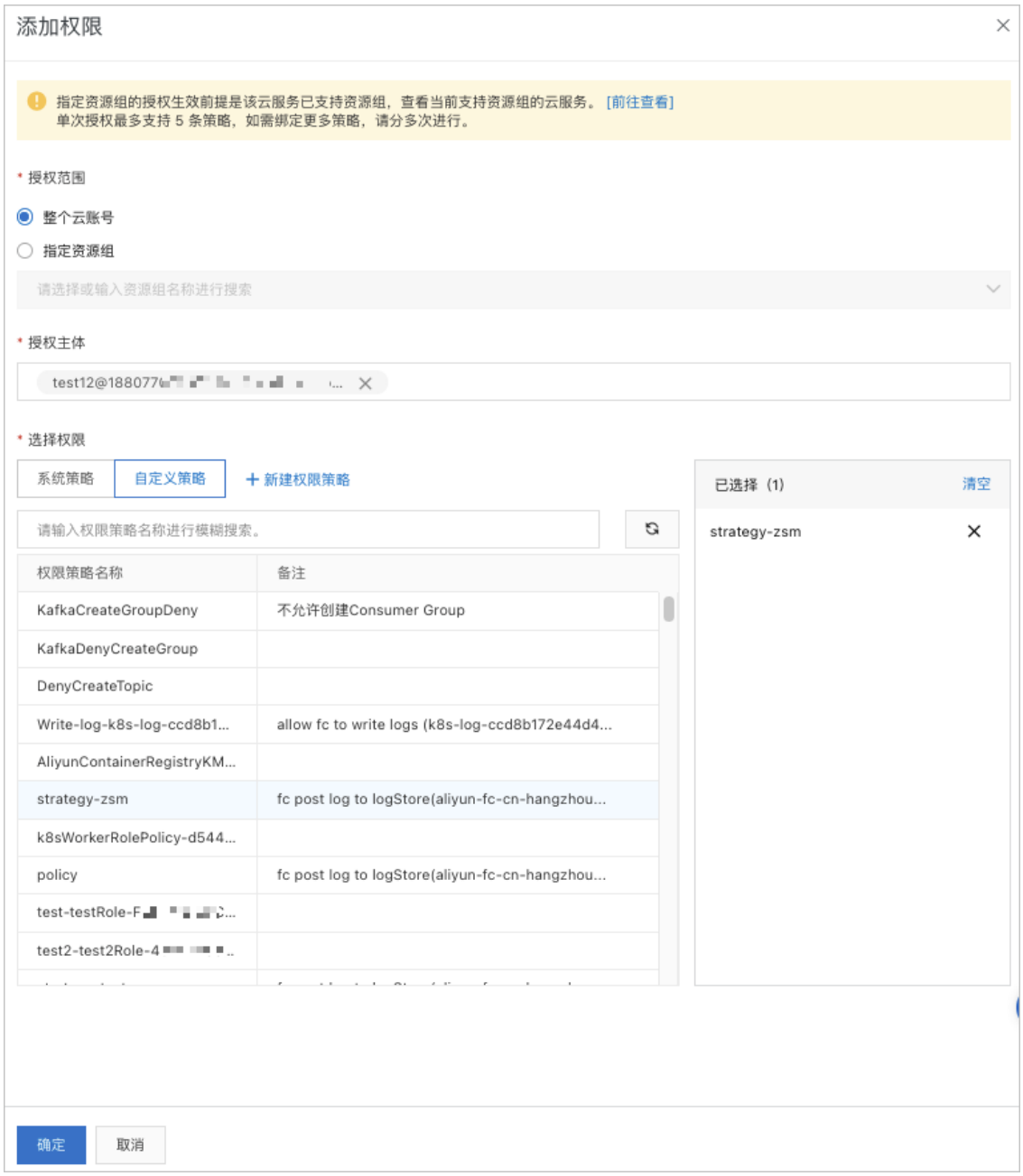
```
{
  "Statement": [
    {
      "Action": "edas:ReadApplication",
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": "edas:ReadApplication",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "edas:tag/Enviroment": ["PROD"]
        }
      }
    }
  ],
  "Version": "1"
}
```

5. 输入策略**名称**和**（可选）备注**，然后在**策略内容**区域检查设置的策略内容，然后单击**确定**。

添加完自定义权限策略后，新增策略出现在权限策略列表。

为RAM用户授权

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏选择**人员管理 > 用户**。
3. 在用户列表页面找到目标RAM用户，单击**操作**列下的**添加权限**。
4. 在**添加权限**面板的**授权范围**区域，单击**云账号全部资源**。
5. 在**添加权限**面板的**选择权限**区域，单击**自定义策略**，在搜索文本框搜索目标策略，然后单击目标策略，再单击**确定**。



6. 在**添加权限**面板，确认授权信息并单击**完成**。

RAM用户访问资源

RAM用户登录EDAS控制台，查看是否能访问目标资源。

7.4. 配置管理

7.4.1. 配置模板管理

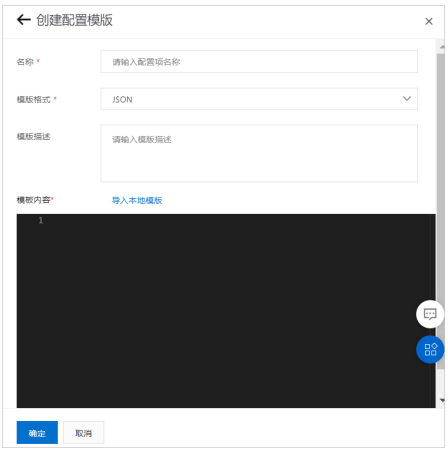
EDAS支持将常用的配置文件存储为配置模板。微服务配置、Kubernetes配置等常用配置都可使用配置模板，方便进行快速配置。

配置模板的使用场景

- ConfigMap配置：支持导入各种格式的配置模板。
- Secret配置：支持导入各种格式的配置模板。

创建模板

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 配置模板**。
3. 在**配置模板**页面单击**创建配置模板**。
4. 在**创建配置模板**面板中设置配置参数，然后单击**确定**。



配置参数说明：

参数	描述
名称	您所创建的模板名称，自定义即可。 **说明** 模板名称是唯一的，不可重复。
模版格式	支持常见文本格式、自定义文本和键值对。常见文本格式包括JSON、XML、YAML、Properties。
模版描述	您可在此添加描述模板用途的信息。
模板内容	支持导入本地模板，文件数据大小不能超过1024 KB。

配置创建成功后，即显示在**配置模板**页面的列表中。

修改配置模板

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 配置模板**。

3. 在**配置模板**页面单击目标模板操作列的**编辑**。
4. 在目标配置模板面板中修改配置参数，然后单击**确定**。

删除配置模板

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 配置模板**。
3. 在**配置模板**页面单击目标模板操作列的**删除**。
4. 在弹出的页面中单击**确定**。

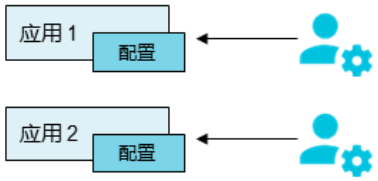
7.4.2. 微服务配置

7.4.2.1. 配置管理概述

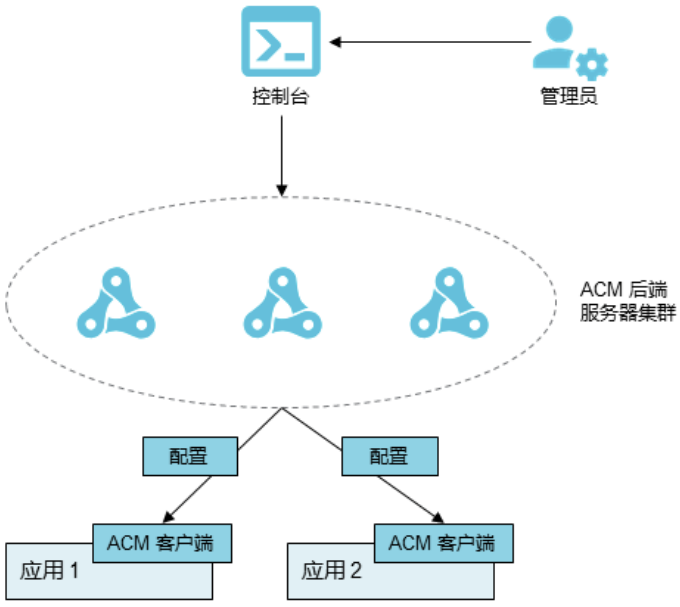
EDAS集成了应用配置管理ACM，您可以在EDAS中集中管理应用配置。

为什么需要应用配置管理ACM

在传统架构中，如需变更配置，通常需要登录服务器并手动修改配置来使配置生效。



在ACM的配置管理场景下，您只需要在EDAS控制台上更改配置，配置信息就会自动被推送到各个服务器中，并在数秒内生效。



关于ACM的更多信息，请参见什么是应用配置管理ACM？。

功能

EDAS通过ACM为应用提供了配置管理的能力。你可以根据实际场景选择相应的配置管理功能。

功能	描述
创建配置	您可以将应用中的变量、参数等从代码中提取出来，并存入一个配置文件，这样在需要更改配置时，只需更改此配置文件即可。
同步配置	您可能需要为应用的多个环境（例如开发环境、测试环境等）同步配置，或者您的应用部署在阿里云的不同地域上，需要跨地域同步配置。您可以创建多个微服务空间，在其中一个空间中创建配置，再将此配置同步其他空间中。
管理配置	创建完配置后，您可以在EDAS控制台搜索、查看、编辑或删除配置。
查看历史版本和回滚配置	如果配置存在问题，您可以查询配置的历史版本，还可以将配置回滚到某个历史版本。
监听查询	修改配置后，查看修改后的配置是否已推送到监听该配置的实例上。
查询配置推送轨迹	修改完配置后，可以根据推送轨迹查看配置是否推送完成。配置修改完发现某个实例不生效，可以根据推送轨迹排查是否推送。

7.4.2.2. 配置管理的基本概念

本文介绍配置管理相关的基本概念。

配置

在系统开发过程中通常会将一些需要变更的参数、变量等从代码中分离出来独立管理，以独立的配置文件的形式存在。目的是让静态的系统工件或者交付物（如WAR，JAR包等）更好地和实际的物理运行环境进行适配。配置管理一般包含在系统部署的过程中，由系统管理员或者运维人员完成这个步骤。配置变更是调整系统运行时行为的有效手段之一。

配置管理

编辑、存储、分发、变更管理、历史版本管理、变更审计等所有与配置相关的活动，统称为配置管理。

动态配置和静态配置

系统配置可以是静态或者动态配置。

- 静态配置：配置的版本与软件本身的版本强绑定。
- 动态配置：在同一个版本软件部署且运行期间，配置可以连续发生多次变更，则称之为动态配置。

例如 `build-version: 1.0.0` 配置与软件版本绑定，则为静态配置；而线程池大小的配置则可以在软件运行期间连续多次变更，则为动态配置。

配置推送

配置管理中，通常需要将配置的变更分发到相关的系统。从分发到配置生效的过程称为配置推送。

推送轨迹

从配置变更、配置推送到配置生效过程的整个轨迹称为推送轨迹。通过查看某个配置的推送轨迹，可以获知一个配置在哪些应用、哪些机器上变更，在哪个时间点生效，产生了哪些影响等。

配置监听

配置监听，是指配置管理组件允许软件系统通过使用SDK等方式向配置管理系统注册监听器（Listener），从而监听并消费该配置的变更。

配置项

配置项是指一个具体的可配置的参数与其值域，通常是param-key=param-value的形式存在。例如配置系统的日志输出级别（ `logLevel=INFO|WARN|ERROR` ）就是一个配置项。

配置集

一组相关或者不相关的配置项的集合称为配置集。通常系统中的一个配置文件就是一个配置集，包含了系统各个方面配置。例如一个配置集可能包含数据源、线程池、日志级别等配置项。

配置集ID（Data ID）

配置集的ID，是配置组织的维度之一。一般通过Data ID来组织、划分系统的配置集。一个系统或者应用可以包含多个配置集，每个配置集可以用有意义的名称来标识这个配置集。Data ID通常采用类Java包命名方式（如 `com.taobao.tc.refund.log.level` ）的命名规则保证全局唯一性。此命名规则非强制。

配置分组（Group）

配置集的分组，是配置组织的维度之一。通常使用一个有意义的字符串来分组配置集，例如Buy，Trade等，用以区分相同Data ID的配置集。创建配置时，如果未填Group名字，则默认用DEFAULT_GROUP代替。Group的常用场景是同一个配置类型用于不同应用或者组件，如database_url配置，MQ_topic配置等。

命名空间（Namespace）

命名空间用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的Group，Data ID的配置。Namespace的常用场景之一是不同环境的配置的区分隔离，如开发测试环境和生产环境的配置隔离等。

配置快照

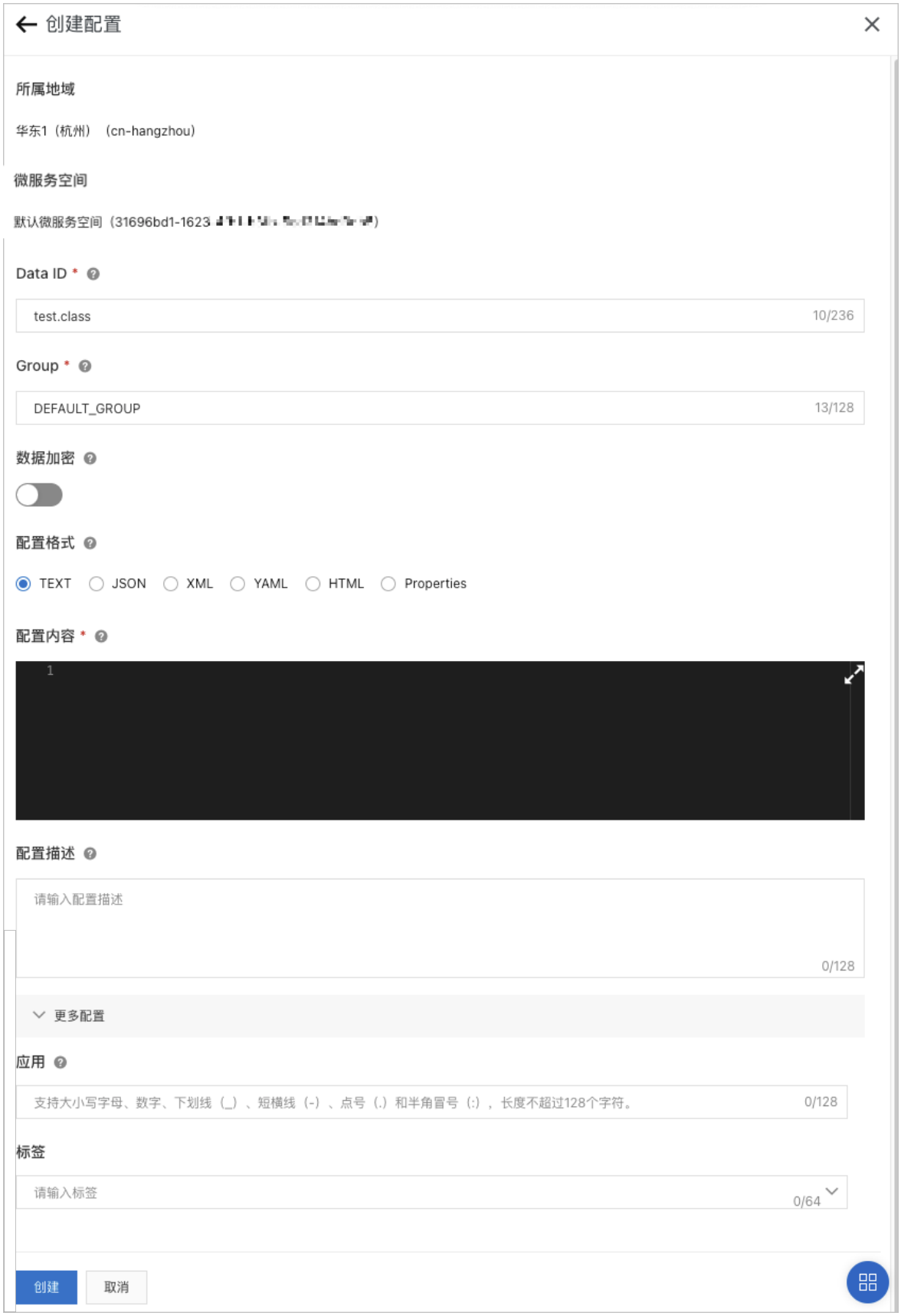
配置管理客户端SDK会在本地生成配置的快照。当客户端无法连接到配置管理服务端时，可以利用快照提供系统的整体容灾能力。配置快照类似于Git中的本地commit的概念，也类似缓存，会在适当的时机更新，但是不会缓存过期（expire）。

7.4.2.3. 创建配置

您可以将应用中的变量、参数等从代码中提取出来，并存入一个配置文件，这样在需要更改配置时，只需更改此配置文件即可。本文介绍如何创建配置文件。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. 在**配置列表**页面顶部菜单栏选择地域，在页面上方选择微服务空间，然后单击**创建配置**。
4. 在**创建配置**面板中设置配置参数，然后单击**创建**。


创建配置
所属地域
华东1（杭州） (cn-hangzhou)
微服务空间
默认微服务空间（31696bd1-1623
Data ID *
test.class
10/236
Group *
DEFAULT_GROUP
13/128
数据加密
配置格式
TEXT
JSON
XML
YAML
HTML
Properties
配置内容 *
1
配置描述
请输入配置描述
0/128
更多配置
应用
支持大小写字母、数字、下划线（_）、短横线（-）、点号（.）和半角冒号（:），长度不超过128个字符。
0/128
标签
请输入标签
0/64
创建
取消

配置参数说明：

参数	描述
所属地域	配置所属地域。在**配置列表**页面选择，此处不可设置。
微服务空间	配置所属微服务空间。在**配置列表**页面选择，此处不可设置。
Data ID	配置ID。建议采用 `package.class` 的命名规范，其中 `class` 部分是具有业务含义的配置名称，例如 `com.foo.bar.log.level` 。 **说明** Data ID在一个Group下必须是唯一的。
Group	配置分组，建议填写产品名或模块名。Group必须是全局唯一的。
数据加密	数据加密包含两种密钥管理服务（KMS）加密方式。 ○ KMS加密：直接调用KMS服务对配置进行加解密，加解密数据的大小不超过6 KB，对特殊符号如and（&）会解密错误，不推荐使用。 ○ KMS AES-128加密：使用KMS的信封加解密方法，配置内容可以超过6 KB，最大不超过100 KB。配置内容的明文数据不会传输到KMS系统，安全性更高，推荐使用。 数据加密的更多操作，请参见创建和使用加密配置。
配置格式	配置内容的数据格式。

配置内容	输入配置的内容，例如：
	`threadPoolSize=5`
	`logLevel=WARN`

配置描述	配置描述信息。
更多配置	○ **应用**：配置归属的应用名。 ○ **标签**：在文本框中输入标签信息，并单击标签选择器。

配置创建成功后，即显示在**配置列表**页面的列表中。

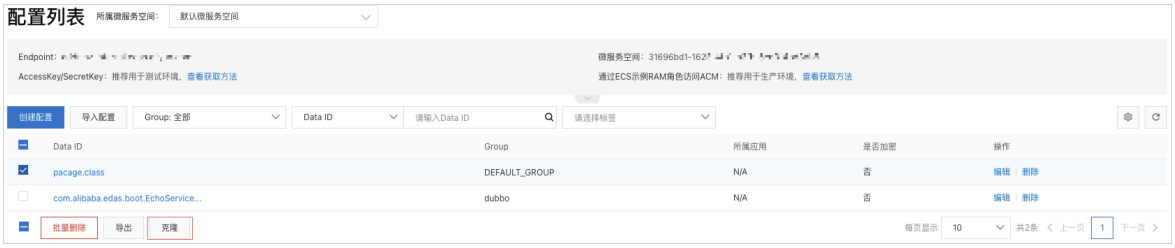
7.4.2.4. 同步配置

您可能需要为应用的多个环境（例如开发环境、测试环境等）同步配置，或者您的应用部署在阿里云的不同地域上，需要跨地域同步配置。您可以创建多个微服务空间，在其中一个空间中创建配置，再将此配置同步到其他空间中。

为同地域内的多个微服务空间同步配置

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. （可选）在**配置列表**页面上方选择微服务空间快速筛选配置，或通过**Group**、**Data ID**、**标签**查询配置。

4. 在**配置列表**页面，选中目标配置，然后在配置列表下方单击**克隆**。

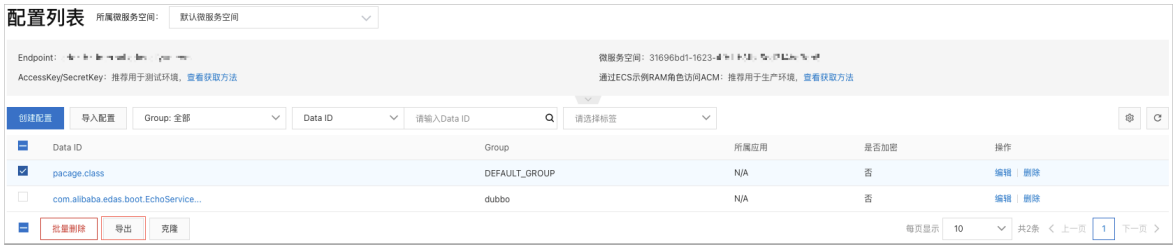


5. 在**克隆配置**对话框中，设置克隆参数，然后单击**克隆**。

参数	描述
目标微服务空间	同步配置的目标微服务空间。
相同配置	遇到相同配置时的处理方式。如果Data ID和Group都相同，则识别为相同配置。 ○ *终止克隆*：终止克隆，此次操作选中的所有配置都不会被同步到目标微服务空间。 ○ *跳过*：跳过重复的配置，继续克隆其他配置。 ○ *覆盖*：用此次选择的配置覆盖目标空间中已有的相同配置。

为不同地域的微服务空间同步配置

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 配置列表**。
3. 在**配置列表**页面顶部菜单栏选择地域，在页面上方选择微服务空间，在列表中选中要导出的配置，然后在下方单击**导出**。



4. 在**导出配置**对话框中确认配置信息，然后单击**导出**。

   配置文件将以ZIP格式下载到本地。

5. 在**配置列表**页面顶部菜单栏选择需要导入配置的地域，在页面上方选择目标微服务空间，然后在页面单击**导入**。

6. 在**导入配置**对话框中上传配置文件，然后单击**导入**。

参数	描述
目标微服务空间	需要导入配置的目标微服务空间。在**配置列表**页面选择，此处不可设置。
相同配置	遇到相同配置时的处理方式。如果Data ID和Group都相同，则识别为相同配置。 ○ *终止导入*：此次操作选中的所有配置都不会被同步到目标微服务空间。 ○ *跳过*：跳过重复的配置，继续克隆其他配置。 ○ *覆盖*：用此次选择的配置覆盖目标空间中已有的相同配置。
配置文件	单击**上传文件**，选择并上传此前导出的配置文件。 **注意** 如果您需要对导入的配置进行修改，请勿直接修改导出在本地的ZIP文件，这样可能会导致导入失败。建议您将配置导入之后再做修改。

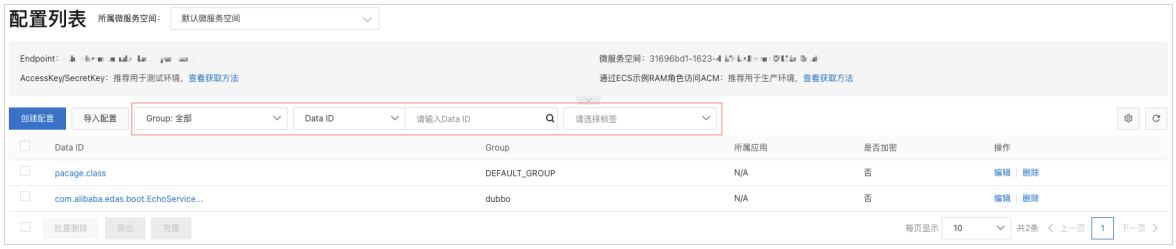
7.4.2.5. 管理配置

创建完配置后，就可以在代码里使用了。由于配置容易发生变化，因此经常需要进行管理操作。本文介绍如何在EDAS控制台查询、编辑和删除配置。

查询配置

EDAS支持通过Data ID、Group ID或其组合查询配置。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. 在**配置列表**页面顶部菜单栏选择地域，在页面上方选择命名空间，然后通过**Group**、**Data ID**、**所属应用**、**标签**查询配置。

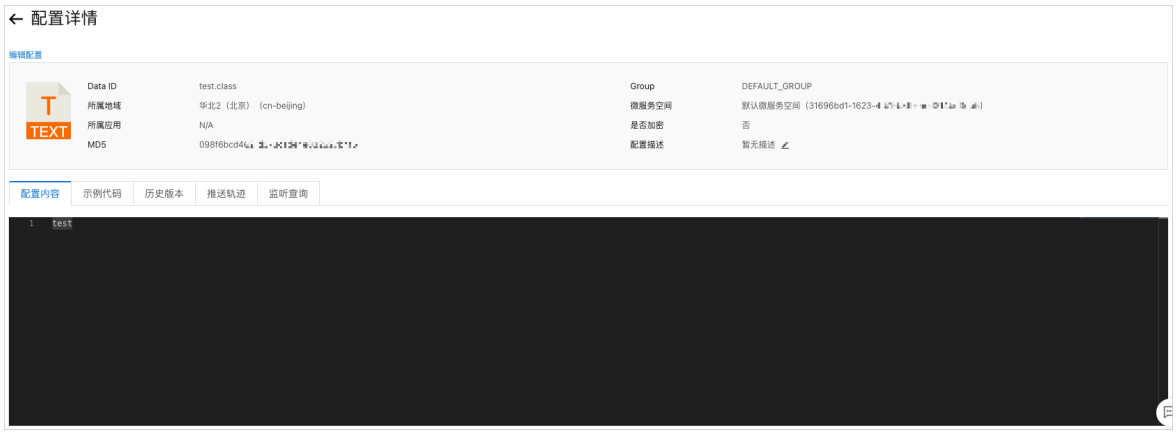


查询到配置之后，可以根据实际需求，完成以下操作。

- 单击配置的**Data ID**来查看配置详情。
- 在配置的**操作**列单击**编辑**来修改配置。
- 在配置的**操作**列单击**删除**来删除配置。
- 选中目标配置，在页面下方单击**导出**来导出配置。
- 选中目标配置，在页面下方单击**克隆**来将配置克隆到同地域的其他命名空间内。

查看配置详情

1. 查询配置，具体操作，请参见查询配置。
2. 在**配置列表**页面单击目标配置的**Data ID**。
3. 在**配置详情**页面查看配置的基本信息、配置内容、示例代码、历史版本、推送轨迹和监听查询信息。



编辑配置

1. 在**配置列表**页面目标配置的**操作**列单击**编辑**。
2. 在**编辑配置**面板中修改配置，然后单击**发布**。

   配置参数，请参见创建配置。**编辑配置**页面中额外提供了以下三个高级特性。

功能	简介	使用说明
格式校验	提供JSON，XML格式等语法校验能力。在配置格式中选择不同的格式，会提供不同格式预发校验，减少语法格式导致的问题。	编辑配置前，在**配置格式**单选框中选择编辑文本的格式。
变更对比	在您修改完配置，提交发布的时候，提供变更对比能力，降低误操作概率。	修改完配置，单击**发布**，弹出配置内容对比框。
Beta发布	对于重要的配置变更，一个变更错误可能导致巨大故障，因此需要将该配置发到几台实例上先验证一下。如果没有问题再全部推送，降低错误变更影响。	选中**Beta发布**，填写需要Beta发布的实例IP（本机测试注意填写公网IP）。

删除配置

1. 在**配置列表**页面目标配置的**操作**列单击**删除**。
2. 在**删除配置**对话框确认删除的配置信息，确认无误后，单击**确定**。

7.4.2.6. 查看历史版本和回滚配置

如果配置存在问题，您可以查询配置的历史版本，还可以将配置回滚到某个历史版本。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. 在左侧导航栏选择**配置管理 > 历史版本**。
4. 在**历史版本**页面选择目标Group，输入待查询配置的Data ID，并单击搜索图标。
   匹配的搜索结果显示在列表中。
5. 在搜索结果中，您可执行以下操作。
   - 单击目标配置**操作**列的**查看**，可以查看该历史配置版本的详细信息。
   - 单击目标配置**操作**列的**回滚**，在弹出的**回滚版本**面板查看配置内容对比，确认回滚则单击**确定**。

> **说明** 配置管理目前只保存30天的变更记录。

7.4.2.7. 监听查询

修改配置后，查看修改后的配置是否已推送到监听该配置的实例上。此查询只对使用了监听配置接口的客户端有效。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. 在配置管理的左侧导航栏，单击**监听查询**。
4. 在**监听查询**页面选择**查询维度**和Group，输入待查询配置的Data ID，并单击搜索图标。

> 说明
> - 如果将查询维度设为**配置**，则表示查询该配置推送到的机器及推送状态。
> - 如果将查询维度设为**IP**，则表示查询该机器监听的所有配置。

7.4.2.8. 查询配置推送轨迹

修改完配置后，可以根据推送轨迹查看配置是否推送完成。若发现某个实例配置不生效，可以根据推送轨迹进行排查。

操作步骤

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > 微服务配置**。
   默认进入配置管理的**配置列表**页面。
3. 在**配置列表**左侧导航栏单击**推送管理**。
4. 在左侧导航栏选择**配置管理 > 推送管理**。
5. 在**推送管理**页面选择**查询维度**和**Group**，输入待查询配置的**Data ID**，并单击搜索图标。

> 说明
> - 如果将查询维度设为**配置**，则表示查询该配置推送到的机器及推送状态。
> - 如果将查询维度设为**IP**，则表示查询该机器监听的所有配置。

7.4.3. Kubernetes配置

7.4.3.1. 管理配置项

您可以使用配置项来保存不需要加密的配置信息，如JVM堆内存、JVM属性参数、Java Agent等参数。本文介绍如何管理配置项。

前提条件

- 在容器服务ACK控制台创建集群。
  - 在EDAS中使用容器服务K8s集群，请在容器服务Kubernetes版控制台创建托管版Kubernetes集群或专有版Kubernetes集群，请参见：
    - 快速创建Kubernetes托管版集群
    - 创建Kubernetes专有版集群
  - 在EDAS中使用Serverless K8s集群，请在容器服务Kubernetes版控制台创建Serverless Kubernetes集群，请参见创建Serverless Kubernetes集群。

- 在EDAS中导入Kubernetes集群。具体操作，请参见在EDAS控制台导入Kubernetes集群。

背景信息

您可以将一些不需要加密的配置信息统一保存到配置项，在创建或者部署应用时可以将配置信息直接注入到容器；如果后续修改了配置项内容，只需要重新部署应用便可生效。

配置项主要有以下两种使用场景：

- 使用配置项定义容器的环境变量。具体操作，请参见配置环境变量。
- 将配置项以文件的形式挂载到容器的指定目录。具体操作，请参见配置挂载。

创建配置项

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**配置项**。
4. 在**配置项**页面顶部菜单栏选择地域。
5. 在**配置项**页面，单击**创建配置项**。
6. 在**创建配置项**面板中，设置配置项的名称、集群等基本参数。

参数	描述
名称	自定义设置配置项名称。支持小写字母、短划线（-）和数字，第一个字符必须是字母、最后一个字符不能是短划线（-），最大长度不超过63个字符。
集群名称	从下拉列表中选择目标Kubernetes集群。

参数	描述
K8s命名空间	K8s Namespace通过将系统内部的对象分配到不同的Namespace中，形成逻辑上分组的不同项目、小组或用户组，便于不同的分组在共享使用整个集群的资源的同时还能被分别管理。 ○ **default**：没有其他命名空间的对象的默认命名空间。 ○ **kube-system**：系统创建的对象的命名空间。 ○ **kube-public**：此命名空间是自动创建的，并且可供所有用户（包括未经过身份验证的用户）读取。 此处以选择**default**为例。

7. 在**创建配置项**面板中，设置配置项映射，然后单击**确定**。

   支持手动添加和文件导入两种设置配置项映射的方式，请根据实际需求选择。在同一个配置项中，支持设置多个映射。

   ○ 手动输入配置项映射。
   单击**添加**，设置配置项的**键**和**值**。

参数	描述
键	配置信息的Key。支持字母、数字、下划线（_）、短划线（-）和半角句号（.）。
值	配置信息的Value。

   ○ 从文件导入配置项映射。
   单击**从文件导入**，从本地选择配置项映射文件。EDAS会自动解析配置项映射文件，以键值对形式展示。
   EDAS会对配置项映射文件进行格式检验，目前支持JSON、YAML和Properties三种类型文件。

   > **说明** 请跟据您的文件类型，设置配置项映射信息。此处仅提供不同文件格式的示例。
   >
   > ■ JSON格式示例
   >
   > ```
   > {
   >   "key1": "value1",
   >   "key2": "value2",
   > }
   > ```
   >
   > ■ YAML格式示例
   >
   > ```
   > key1: value1
   > key2: value2
   > ```
   >
   > ■ Properties格式示例
   >
   > ```
   > key1=value1
   > key2=value2
   > ```

查看配置项

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**配置项**。
4. 在**配置项**页面顶部菜单栏选择地域，在页面中选择微服务空间。

5. 在**配置项**页面，单击目标配置项后的**详情**。

   您可以通过**配置项名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标配置项。

6. 在配置项的详情页面，查看该配置项的基本信息，以及配置项包含的数据信息。

修改配置项

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**配置项**。
4. 在**配置项**页面顶部菜单栏选择地域，在页面中选择微服务空间。
5. 在**配置项**页面找到目标配置项，单击右侧的**编辑**。

   您可以通过**配置项名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标配置项。

6. 在编辑面板中，修改配置项的映射名称和值，然后单击**确定**。

   > **说明** 如果已经有应用使用该配置项，请在编辑完成后重新部署应用，以保证编辑后的配置项信息在应用中生效。

删除配置项

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**配置项**。
4. 在**配置项**页面顶部菜单栏选择地域，在页面中选择微服务空间。
5. 在**配置项**页面找到目标配置项，单击右侧的**删除**。

   您可以通过**配置项名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标配置项。

6. 在确认删除对话框，单击**确定**。

   > **说明** 如果已经有应用使用该配置项，不建议删除配置项。如果您删除了已被应用使用的配置项，则应用重启或重新部署后无法正常启动。

7.4.3.2. 管理保密字典

如果您需要在EDAS K8s环境中使用一些敏感的配置，例如密码、证书等信息时，建议使用保密字典（Secret）。本文介绍如何管理保密字典。

前提条件

- 在容器服务ACK控制台创建集群。
  - 在EDAS中使用容器服务K8s集群，请在容器服务Kubernetes版控制台创建托管版Kubernetes集群或专有版Kubernetes集群，请参见：
    - 快速创建Kubernetes托管版集群
    - 创建Kubernetes专有版集群
  - 在EDAS中使用Serverless K8s集群，请在容器服务Kubernetes版控制台创建Serverless Kubernetes集群，请参见创建Serverless Kubernetes集群。
- 在EDAS中导入Kubernetes集群。具体操作，请参见在EDAS控制台导入Kubernetes集群。

背景信息

您可以将一些敏感信息（如密码、证书等信息）统一保存到保密字典，在创建或者部署应用时可以将配置信息直接注入到容器；如果后续修改了保密字典内容，只需要重新部署应用便可生效。

保密字典主要有以下三种使用场景：

- 使用保密字典定义容器的环境变量。具体操作，请参见配置环境变量。
- 将保密字典以文件的形式挂载到容器的指定目录。具体操作，请参见配置挂载。
- 在保密字典中的HTTPS证书信息可以直接用于应用路由Ingress。具体操作，请参见添加应用路由Ingress。

创建保密字典

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**保密字典**。
4. 在**保密字典**页面顶部菜单栏选择地域。
5. 在**保密字典**页面，单击**创建保密字典**。
6. 在**创建保密字典**面板中，设置保密字典参数，然后单击**确定**。

参数	描述
保密字典名称	自定义设置保密字典名称。支持小写字母、短划线（-）和数字，第一个字符必须是字母，最后一个字符不能是短划线（-）。
集群名称	从下拉列表中选择目标Kubernetes集群。
K8s命名空间	K8s Namespace通过将系统内部的对象分配到不同的Namespace中，形成逻辑上分组的不同项目、小组或用户组，便于不同的分组在共享使用整个集群的资源的同时还能被分别管理。 ○ **default**：没有其他命名空间的对象的默认命名空间。 ○ **kube-system**：系统创建的对象的命名空间。 ○ **kube-public**：此命名空间是自动创建的，并且可供所有用户（包括未经过身份验证的用户）读取。
类型	选择创建的保密字典类型，目前支持创建**Opaque**和**TLS证书**两类。 ○ **Opaque**：用户自定义的任意数据。当您选择此项时，会呈现**Base64编码数据**复选框。 当您想上传二进制数据转化成Base64编码的文本时，可选中**Base64编码数据**复选框。勾选后，保密字典须配置已经过Base64编码处理的数据，EDAS将不再对数据进行编码。 ○ **TLS证书**：用于保存TLS证书及其相关密钥。主要用在应用路由Ingress场景，支持将外部HTTPS请求路由到内部Service的路由规则集合。
Opaque	创建**Opaque**类型的保密字典，需要设置以下参数： ○ 映射参数： ■ **键**：敏感信息的Key。支持字母、数字、短划线（-）、下划线（_）和半角句号（.）。 ■ **值**：敏感信息的Value。 您也可以单击**导入配置**，直接从本地导入配置文件，数据值大小不能超过1024K，支持 `json`、`yaml`、`properties` 类型。 ○ 当您想上传二进制数据转化成Base64编码的文本时，可选中**Base64编码数据**复选框。勾选后，保密字典须配置已经过Base64编码处理的数据，EDAS将不再对数据进行编码。
TLS证书	创建**TLS证书**类型的保密字典，支持**创建自签名证书**和手动创建。 ○ 手动创建：需要手动输入**Cert**（TLS证书的公钥）和**Key**（TLS证书的私钥）。 ○ 创建自签名证书：通过配置域名、密钥长度、证书起始时间、证书失效时间生成证书。

查看保密字典

1. 登录EDAS控制台。

2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**保密字典**。
4. 在**保密字典**页面顶部菜单栏选择地域。
5. 在**保密字典**页面，单击目标保密字典后的**详情**。

   您可以通过**保密字典名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标保密字典。
6. 在保密字典的详情页面，查看该保密字典的基本信息，以及保密字典包含的数据信息。

   对于TLS类型证书可在详情页查看证书的详细信息，包括证书关联域名、证书状态、证书服务提供商等。

修改保密字典

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**保密字典**。
4. 在**保密字典**页面顶部菜单栏选择地域。
5. 在**保密字典**页面找到目标配置项，单击右侧的**编辑**。

   您可以通过**保密字典名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标保密字典。
6. 在编辑面板中，修改保密字典的映射名称和值，然后单击**确定**。

> **说明** 如果已经有应用使用该保密字典，请在编辑完成后重新部署应用，以保证编辑后的保密字典信息在应用中生效。

查看关联路由

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**保密字典**。
4. 在**保密字典**页面顶部菜单栏选择地域。
5. 在**保密字典**页面找到目标配置项，单击右侧的**查看关联路由**。

   您可以通过**保密字典名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标保密字典。
6. 在弹出的页面查看字典关联的应用路由，单击目标应用路由名称可查看应用路由基本信息。

删除保密字典

1. 登录EDAS控制台。
2. 在左侧导航栏选择**配置管理 > Kubernetes配置**。
3. 在**Kubernetes配置**的左侧导航栏单击**保密字典**。
4. 在**保密字典**页面顶部菜单栏选择地域。
5. 在**保密字典**页面找到目标配置项，单击右侧的**删除**。

   您可以通过**保密字典名称**、**集群名称**、**集群ID**和**K8s命名空间**筛选目标保密字典。
6. 在确认删除对话框，单击**确定**。

> **说明** 如果已经有应用使用该保密字典，建议解除使用关系后再进行删除。

7.5. 查看操作日志

查看操作日志

您可以在EDAS控制台上查看操作日志。如果当前账号为主账号，则可以查询当前账号及其所有子账号在控制台上的操作日志；如果当前账号为某个主账号的子账号，则可以查询该子账号在控制台上的操作日志。

查看操作日志详情

为了便于查看，EDAS对操作进行了分类，您可以选择不同操作类型以及具体操作查看操作日志详情。

1. 登录EDAS控制台。
2. 在左侧导航栏选择**系统管理 > 操作日志**。
3. 在**操作日志**页面设置查询条件，然后单击**搜索**。

   - **应用名称**：默认为全部应用。也可以在下拉列表中选择具体应用。
   - **时间范围**：选择开始时间和结束时间，可以精确到秒。
   - **操作人**：请输入完整账号，不支持模糊匹配。
   - **选择分类**：选择操作分类和具体的操作。
4. 在搜索出来的操作列表的**操作**列单击**查看详情**。

   您将进入**变更记录**页面查看变更详情和操作日志。

7.6. 系统管理常见问题

7.6.1. 如何设置RAM用户的联系方式？

在创建RAM用户时设置的联系方式目前无法在EDAS中使用。如果把RAM用户设为报警联系人，请按以下步骤设置联系方式。

1. 使用RAM用户登录EDAS控制台。
2. 在**系统管理 > 个人资料**中可以更新个人资料，包括联系人名称、电话、邮箱。
3. 单击**保存**后，即可将RAM用户信息保存在EDAS中。

8.SDK参考

8.1. Java SDK接入指南

您可以使用EDAS提供的Java SDK进行API调用。

前提条件

安装Java SDK必须用1.6或更高版本的JDK。

获取Java SDK

您可以通过以下两种方式获取Java SDK：

- **通过Maven直接获取（联网环境下推荐）**
  打开Maven项目下的*pom.xml*文件，添加aliyun-java-sdk-core和aliyun-java-sdk-edas依赖。

```
<dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-core</artifactId>
   <version>4.5.0</version>
</dependency>
<dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-edas</artifactId>
   <version>3.18.0</version>
</dependency>
```

- **离线导入安装 （无互联网连接的环境）**
  首先需要找到一个联网的环境将Java SDK相关的Alibaba Cloud SDK for Java和Alibaba Cloud EDAS SDK for Java的JAR包文件下载下来，然后拷贝到无互联网连接的环境，并添加到项目工程中。

使用Java SDK调用API

实际使用时，请替换以下示例中的aliyun_user_ak、aliyun_user_sk和region_id为您实际的公共参数值，相关公共参数请参见API调用公共参数 。

```
import java.util.List;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.edas.model.v20170801.ListApplicationRequest;
import com.aliyuncs.edas.model.v20170801.ListApplicationResponse;
import com.aliyuncs.edas.model.v20170801.ListDeployGroupRequest;
import com.aliyuncs.edas.model.v20170801.ListDeployGroupResponse;
public class ListApplicationsSimpleInfo {
    public static void main(String args[]){
        String aliyun_user_ak = "yourAccessKeyId";//阿里账号或RAM用户的AccessKey ID。
        String aliyun_user_sk = "yourAccessKeySecret";//阿里云账号或RAM用户的AccessKey Secret。
        String region_id = "cn-hangzhou";//应用所在地域ID。
        DefaultProfile defaultProfile = DefaultProfile.getProfile(region_id, aliyun_user_ak, aliyun_
user_sk);
        DefaultAcsClient defaultAcsClient = new DefaultAcsClient(defaultProfile);
        ListApplicationRequest applist_req = new ListApplicationRequest();
        try {
            ListApplicationResponse applist_resp = defaultAcsClient.getAcsResponse(applist_req);
            if(applist_resp.getCode() == 200){
                List<ListApplicationResponse.Application> applist = applist_resp.getApplicationList(
);
                if(applist != null && applist.size() > 0){
                    for(ListApplicationResponse.Application app : applist){
                        String app_name = app.getName();
                        String app_id = app.getAppId();
                        System.out.println("应用名称 : " + app_name + ", 应用ID : " + app_id);
                        ListDeployGroupRequest dglist_req = new ListDeployGroupRequest();
                        dglist_req.setAppId(app_id);
                        ListDeployGroupResponse dglist_resp = defaultAcsClient.getAcsResponse(dglist
_req);
                        if(dglist_resp.getCode() == 200){
                            List<ListDeployGroupResponse.DeployGroup> dglist = dglist_resp.getDeploy
GroupList();
                            for(ListDeployGroupResponse.DeployGroup dg : dglist){
                                String dg_name = dg.getGroupName();
                                if("_DEFAULT_GROUP".equals(dg_name)){
                                    dg_name = "默认分组";
                                }
                                String dg_id = dg.getGroupId();
                                System.out.println("\t分组名 : " + dg_name + ", 分组Id : " + dg_id);
                            }
                        }
                    }
                } else {
                    System.out.println("获取到的应用列表为空，请检查上面设置的region_id中是否存在应用。");
                }
            } else {
                // 打印错误原因
                System.out.println("API调用返回异常!\nMessage: " + applist_resp.getMessage() + "\nReq
uestId: " + applist_resp.getRequestId());
            }
        } catch (ClientException e) {
            e.printStackTrace();
        }
    }
}
```

> **说明**
>
> - EDAS Java SDK的API接口都是以<接口名>Request和<接口名>Response成对出现的，例如：ListApplicationRequest和ListApplicationResponse，ListDeployGroupRequest和ListDeployGroupResponse 。<接口名>Request 用于构造接口的请求，为该接口添加相关参数，然后用 ACS Client 获取一个 <接口名>Response 对象来承接该接口的返回结果。
> - 当aliyun-java-sdk-core为4.4.3及其以上版本，并且aliyun-java-sdk-edas为2.52.1及其以上版本时，EDAS Java SDK可根据*region_id*自动找到Endpoint domain。

8.2. Python SDK接入指南

你可以使用EDAS提供的Python SDK进行API调用。

获取Python SDK

Python 2.7.x以上版本。

- **使用pip快速安装（联网环境下推荐）**

```
root# pip install -U aliyun-python-sdk-core
root# pip install -U aliyun-python-sdk-edas
```

> **说明** root#表示在Linux环境下需要用root用户执行命令，macOS环境下请在命令前添加sudo来执行上面这两条命令，建议每隔2~3个月就执行上面的命令更新上面的2个SDK包的版本。

- **离线安装（无互联网连接的环境）**
  i. 在联网环境访问阿里云开放平台，根据页面提示下载Python SDK核心库。
  ii. 将得到的压缩包上传至需要运行Python API调用程序的目标主机中并解压，然后依次进入aliyun-python-sdk-core和aliyun-python-sdk-edas子目录，分别执行以下命令安装aliyun-python-sdk-core和aliyun-python-sdk-edas这两个Python SDK：

```
root# python setup.py install
```

使用Python SDK调用API

实际使用时，请替换下面示例中的aliyun_user_ak、aliyun_user_sk、region_id这 3 个公共参数及其他必要的参数，公共参数的详细信息请参见API调用公共参数 。

```
#!/usr/bin/env python
# -*- coding=UTF-8 -*-
import sys, json
from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.profile import region_provider
from aliyunsdkedas.request.v20170801.ListApplicationRequest import ListApplicationRequest
from aliyunsdkedas.request.v20170801.ListDeployGroupRequest import ListDeployGroupRequest
if __name__ == '__main__':
    #请填写阿里云账号或RAM用户的AccessKey ID.
    aliyun_user_ak = 'LTAIPQxxxxxxxxxx'
    #请填写阿里云账号或RAM用户的AccessKey Secret.
    aliyun_user_sk = 'W75qdr9ORkxxxxxxxxxxxxxx'
    #请填写要执行API调用的应用及ECS、SLB、VPC等资源所在地域ID。
    region_id = 'cn-shanghai'
    client = AcsClient(ak=aliyun_user_ak, secret=aliyun_user_sk, region_id=region_id, timeout=300)
    applist_req = ListApplicationRequest()
    applist_resp = json.loads(client.do_action_with_exception(applist_req))
    if applist_resp['Code'] == 200:
        applist = applist_resp['ApplicationList']['Application']
        for app in applist:
            app_name = app['Name']
            app_id = app['AppId']
            print u'应用名称 : ' + app_name + u', 应用Id : ' + app_id
            dglist_req = ListDeployGroupRequest()
            dglist_req.set_AppId(app_id)
            dglist_resp = json.loads(client.do_action_with_exception(dglist_req))
            if dglist_resp['Code'] == 200:
                dglist = dglist_resp['DeployGroupList']['DeployGroup']
                for dg in dglist:
                    dg_name = dg['GroupName']
                    if dg_name == '_DEFAULT_GROUP':
                        dg_name = u'默认分组'
                    dg_id = dg['GroupId']
                    print u'\t分组名: ' + dg_name + u', 分组ID: ' + dg_id
            else:
                print u'获取' + app_name + u'应用的分组列表失败.'
    else:
        print u'获取应用列表失败.'
```

说明

- 从aliyun-python-sdk-core 2.13.9+和aliyun-python-sdk-edas 2.52.1+版本开始，使用EDAS POP API的Python SDK时，不再需要在代码中设置product_name (Edas) 和region_domain（例如：edas.cn-shanghai.aliyuncs.com）这两个参数，aliyun-python-sdk-core会自动根据region_id的值自动解析出该region_id对应的endpoint domain，想体验这个功能，请执行下面的操作将aliyun-python-sdk-core和aliyun-python-sdk-edas版本升级到最新。

  ```
  root# pip install -U aliyun-python-sdk-core
  root# pip install -U aliyun-python-sdk-edas
  root# pip list 2>/dev/null | grep -E "aliyun-python-sdk-core|aliyun-python-sdk-edas"
  aliyun-python-sdk-core        2.13.9
  aliyun-python-sdk-edas        2.52.1
  ```

- 该示例在Python 2.7.x版本测试可用。使用Python 3.x版本时，请注意兼容性。

更多信息

- API调用公共参数
- Java SDK接入指南
- CLI接入指南
- OpenAPI开发者门户

8.3. CLI接入指南

您可以使用阿里云CLI 接入EDAS的API，完成日常的管理操作。

安装CLI

安装CLI的步骤，请参见：

- 在Windows上安装阿里云CLI
- 在Linux上安装阿里云CLI
- 在macOS上安装阿里云CLI

配置CLI

配置CLI的步骤，请参见配置CLI。

使用CLI调用API

CLI的基本使用方法，请参见使用CLI调用API。在使用CLI调用EDAS的API前，您还需要了解EDAS的API列表及各API的请求参数、返回参数等信息。

> **说明** 对于API中不同类型的字段，请遵循阿里云CLI的参数格式要求。更多信息，请参见参数格式说明。

下面以几个样例说明如何使用CLI调用API。

- 创建命名空间

```
aliyun edas InsertOrUpdateRegion --RegionTag cn-beijing:testheng --RegionName testheng --region cn
-beijing --endpoint "edas.cn-beijing.aliyuncs.com"
```

- 查询地域列表

```
aliyun edas ListAliyunRegion
```

- 删除命名空间下的集群

```
aliyun edas DeleteCluster --ClusterId f8b3014e-0f61-493a-a602-6f9b63ba**** --logicalRegionId cn-be
ijing:docNamespace3
```

使用建议

当需要使用CLI调用API完成相对复杂的任务时，建议将CLI整理成Shell脚本，然后执行，可以提高效率。使用CLI调用API的示例场景，请参见使用CLI快速部署应用至ECS集群。

更多信息

- API调用公共参数
- Java SDK接入指南
- Python SDK接入指南
- OpenAPI开发者门户

9.联系我们

如果您在使用企业级分布式应用服务EDAS（Enterprise Distributed Application Service）的过程中有任何疑问或建议，请提交工单，或联系我们的支持人员协助处理，同时也欢迎使用钉钉搜索钉钉群号加入钉钉群进行反馈。

EDAS Spring Cloud和Dubbo交流群

如果您在部署Spring Cloud或Dubbo应用至EDAS的过程中有任何疑问或建议，请提交工单，或使用钉钉搜索钉钉群号 `31723701` 加入钉钉群进行反馈。

EDAS容器服务K8s和Serverless K8s交流群

如果您在EDAS中使用容器服务K8s集群和Serverless K8s集群过程中有任何疑问或建议，请提交工单，或使用钉钉搜索钉钉群号 `23197114` 加入钉钉群进行反馈。

EDAS多语言应用交流群

如果您在部署EDAS多语言微服务应用过程中有任何疑问或建议，请提交工单，或使用钉钉搜索钉钉群号 `23307994` 加入钉钉群进行反馈。

EDAS开发者工具交流群

如果您在EDAS中使用开发者工具的过程中有任何疑问或建议，请提交工单，或使用钉钉搜索钉钉群号 `34556175` 加入钉钉群进行反馈。

EDAS应用创建交流群

如果您在体验EDAS的应用创建过程中有任何疑问或建议，请提交工单，或使用钉钉搜索钉钉群号 `21958624` 加入钉钉群进行反馈。