阿里云
分布式任务调度 SchedulerX

高级特性

文档版本：20200609

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. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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
	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。	 **禁止：** 重置操作将丢失用户配置数据。
	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。
	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。
	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。	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 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。

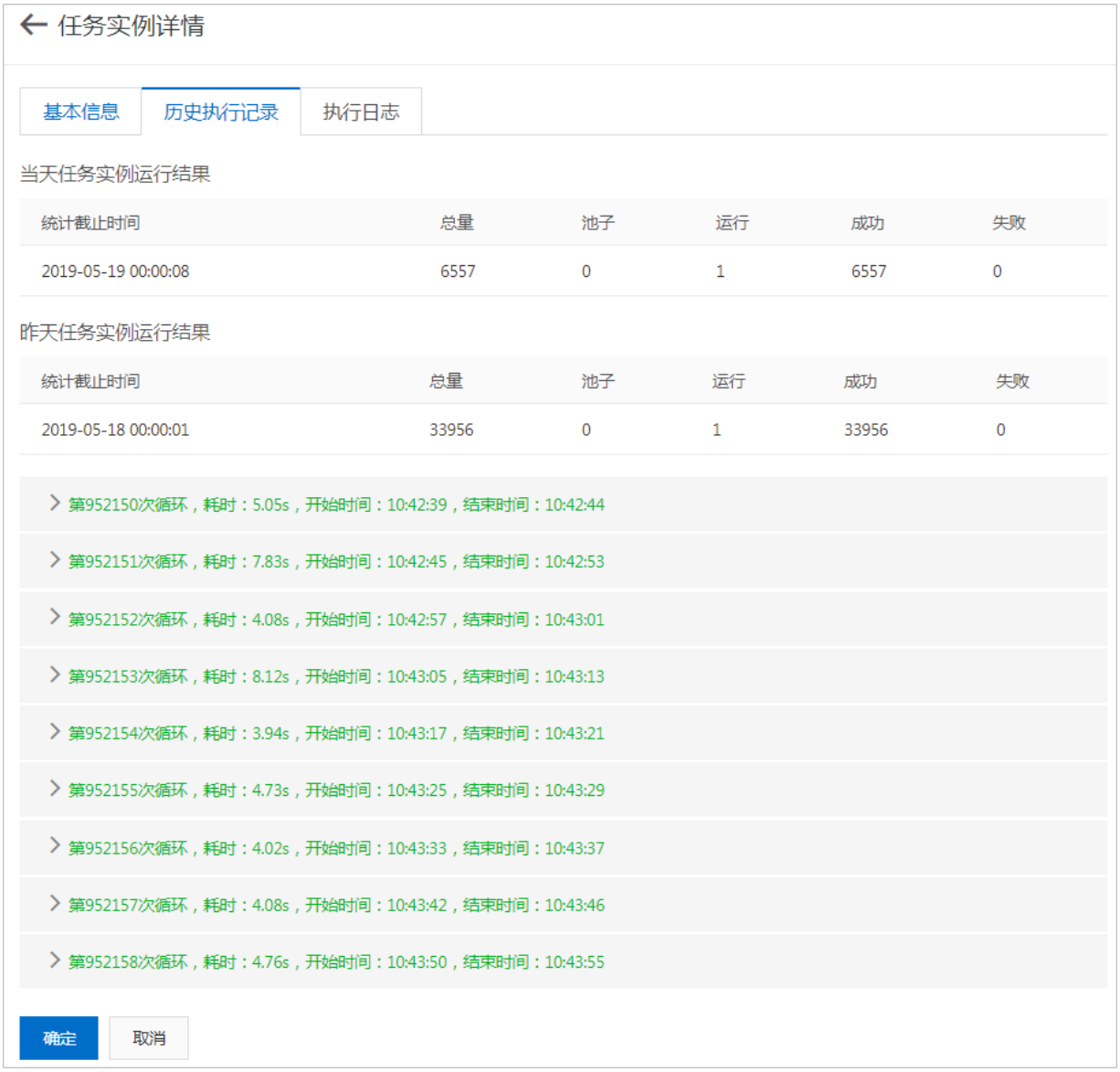
由于秒级调度属于定时调度，所以在定时配置步骤中请将**时间类型**设置为second_delay，并将**固定延迟**设置为50（秒）。创建调度任务的操作步骤请参见创建调度任务。

查看秒级调度任务详情。

查看任务实例详情的步骤请参见#unique_4。

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。

2 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#unique_6。

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));
}
}
```

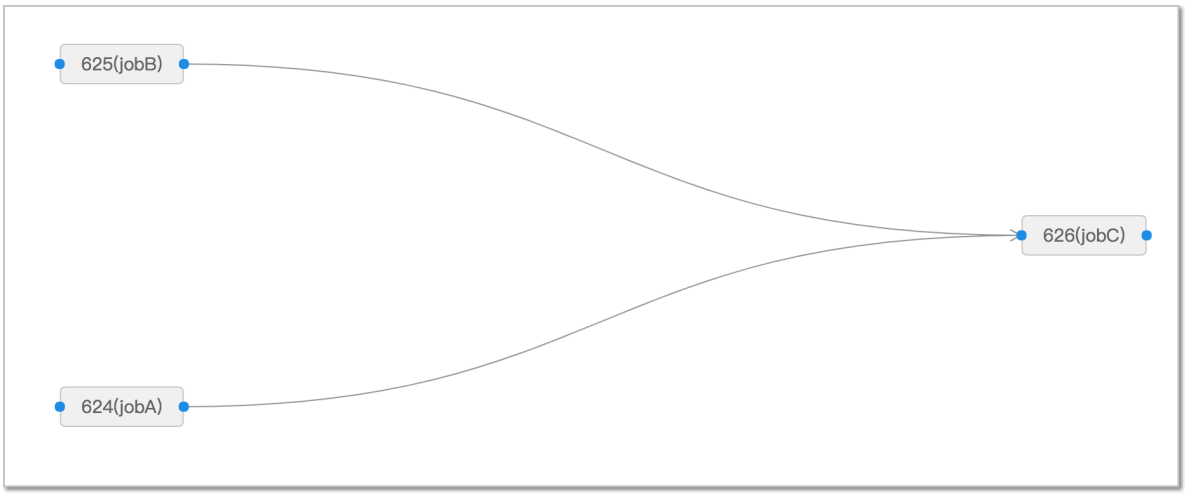
   - JobProcessor C

```
@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，详情请参见#unique_7。
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#unique_8/unique_8_Connect_42_section_p0t_h1h_qkp和#unique_9/unique_9_Connect_42_section_ww9_0re_gta。
4. 创建工作流，并导入这3个调度任务，详情请参见#unique_10/unique_10_Connect_42_section_2b5_ckm_dog。

   创建好的工作流如下图所示：



5. 在**流程管理**页面单击**更多**，然后在下拉列表中选择**运行一次**。

预期结果

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
```

3 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。

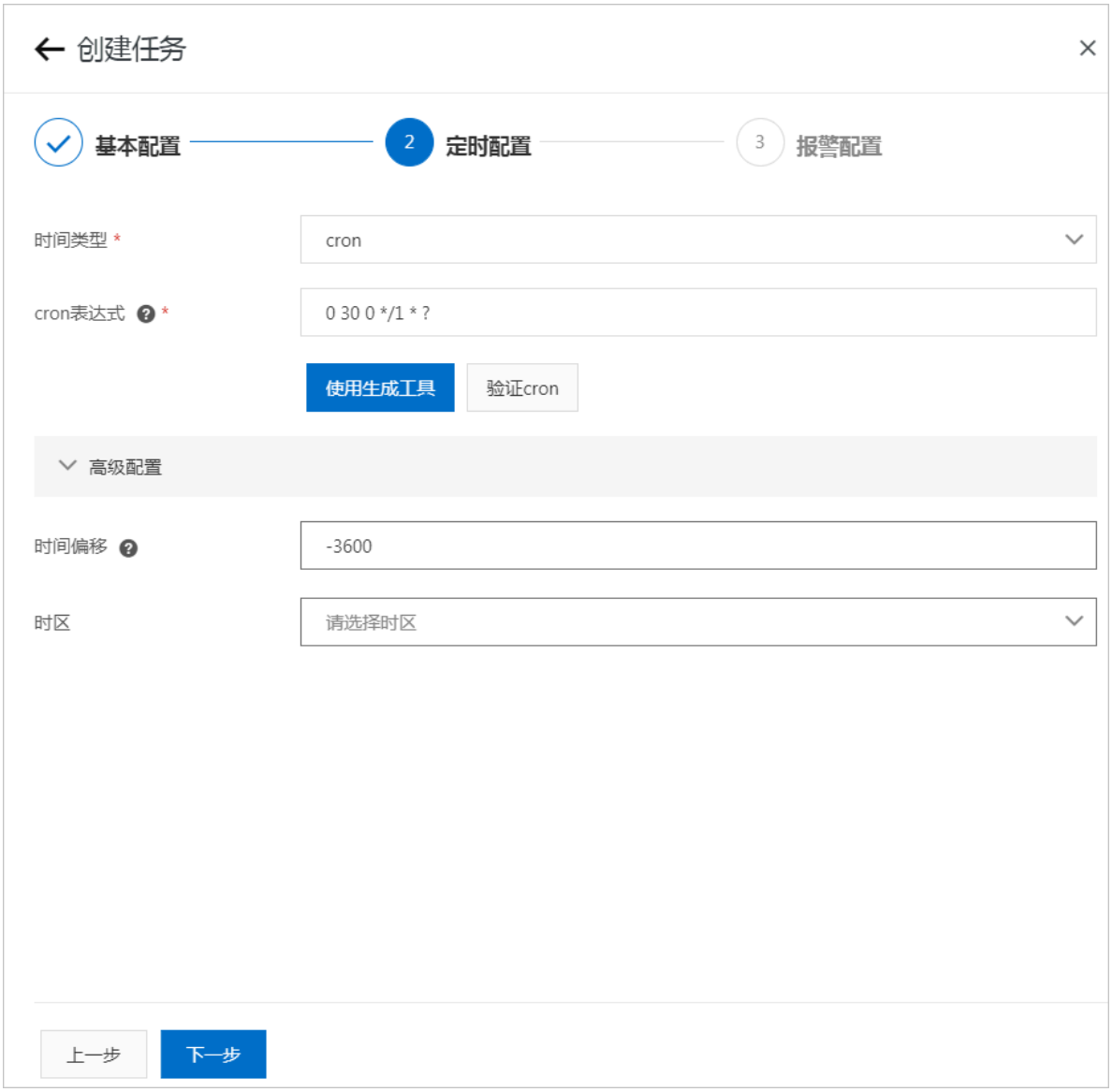
1. 在客户端中接入SchedulerX，详情请参见快速入门章节，并实现数据时间。

```
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#unique_9/unique_9_Connect_42_section_ww9_0re_gta。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。

4 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**页面任务列表的**操作**列单击 图标，在弹出的菜单中单击**重刷任务**。
2. 在**重刷任务**页面设置**起止时间**和**数据时间**，单击**确定**。
   - **起止日期**：指定重刷的时间区间，当前只支持Cron表达式任务。
   - **数据时间**：指定重刷时间区间内的重刷时间。

   示例

   Cron表达式为`0 0 2 * * ?`，表示每天2点运行，这个时候重刷配置如下：

   - 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