

# SOFAStack

## 任务调度 技术白皮书

产品版本：AntStack Plus 1.11.0

文档版本：20220929

# 法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

未经蚂蚁集团事先书面许可，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。

## 商标声明

 蚂蚁集团  
ANT GROUP 及其他蚂蚁集团相关的商标均为蚂蚁集团所有。本文档涉及的第三方的注册商标，依法由权利人所有。

## 免责声明

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。蚂蚁集团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，并在蚂蚁集团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蚂蚁集团授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

# 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 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

# 目录

1.产品简介	05
2.产品优势	06
3.产品架构	07
4.性能指标	10
5.功能原理	14
5.1. 简单任务	14
5.2. 集群任务	14
5.3. 拓扑任务	16
6.单元化能力	18
7.附录：基础术语	19

# 1. 产品简介

任务调度 TS (Task Scheduler) 提供分布式任务调度框架, 实现任务的分布式处理, 并能规范化、自动化、可视化和集中化对金融企业不同业务系统的任务进行统一的调度和全方位监控运维管理, 达到所有任务有序、高效运行的目的, 极大降低开发和运维的成本。

任务调度产品提供多种任务类型满足不同场景需求, 进行可靠的自动化任务调度:

- **简单任务**

简单的 RPC 任务。适用于业务逻辑简单的场景, 当需要并发执行并且不关注资源利用率时, 可以使用简单任务的分片功能; 当需要分成若干步骤按顺序执行时, 可以使用简单任务的分步功能。分片数量在任务设置时指定。

- **集群任务**

支持对数据进行自定义维度的分片, 目前最多支持两层拆分, 可以充分利用集群的所有机器, 当数据量较大时, 多层调度任务可以更快的处理完数据。支持动态分片, 根据业务需求将任务拆分多个分片, 充分利用业务方集群资源, 提供并行处理能力。

- **消息任务**

一种消息队列的消息类型, 由消息主题和消息事件码作为唯一标识, 需要配合消息队列产品使用。

- **拓扑任务**

一种特殊的任务, 是通过事件触发的任务集合。拓扑任务中的任务执行流程均起始于开始节点, 终止于结束节点, 任务的执行流程形成一张有向无环图。

## 2. 产品优势

### 金融级高可用

支持机房级容灾，任何单机或单机房故障都不影响当前任务的执行，保障业务的正常执行。

### 高可靠高扩展

支持无限水平扩展，无性能容量瓶颈，监控任务的执行情况，可以及时发现执行异常或没有执行的任务，通过可配置的异常处理策略对异常任务进行补偿。

### 高性能

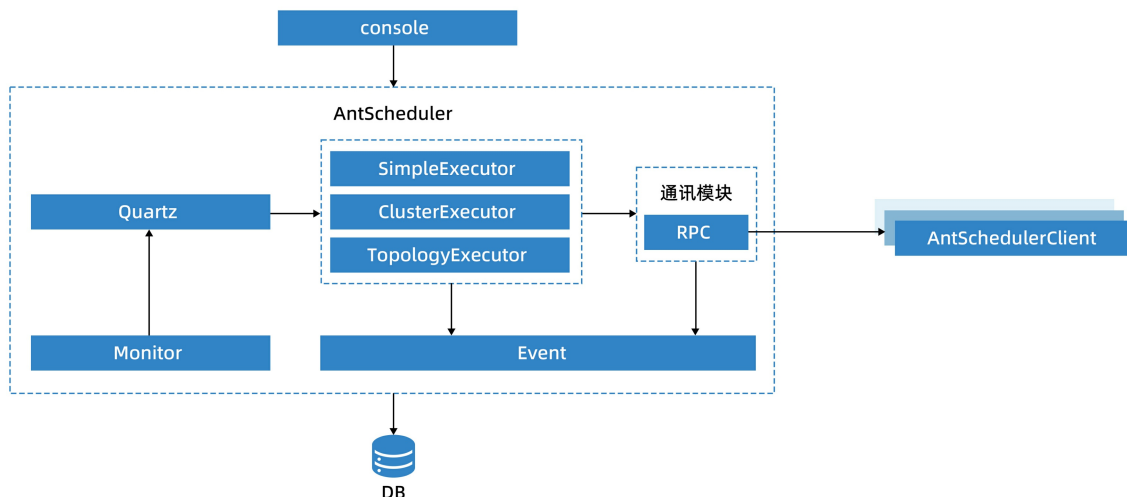
通过分布式以及优化的底层架构、支持多层调度模式、可进行无限拆分、多线程并行处理，显著提升大数据量的批任务处理的性能。

### 可视化集中式管理

通过简易操作的可视化集中式管理平台可对上万个任务节点进行集中化管理，简化运维管理操作，提高处理效率。

## 3. 产品架构

### 系统架构



任务调度由以下三个部分组成：

- **antschedulerconsole**：控制面，是一个可视化集中管理平台。方便管理任务、简化运维操作。
- **antscheduler**：任务调度核心，负责任务的调度和集群间任务分配等功能。
- **client**：客户端，业务方通过集成 client，获取 RPC 和 集群任务调度能力。

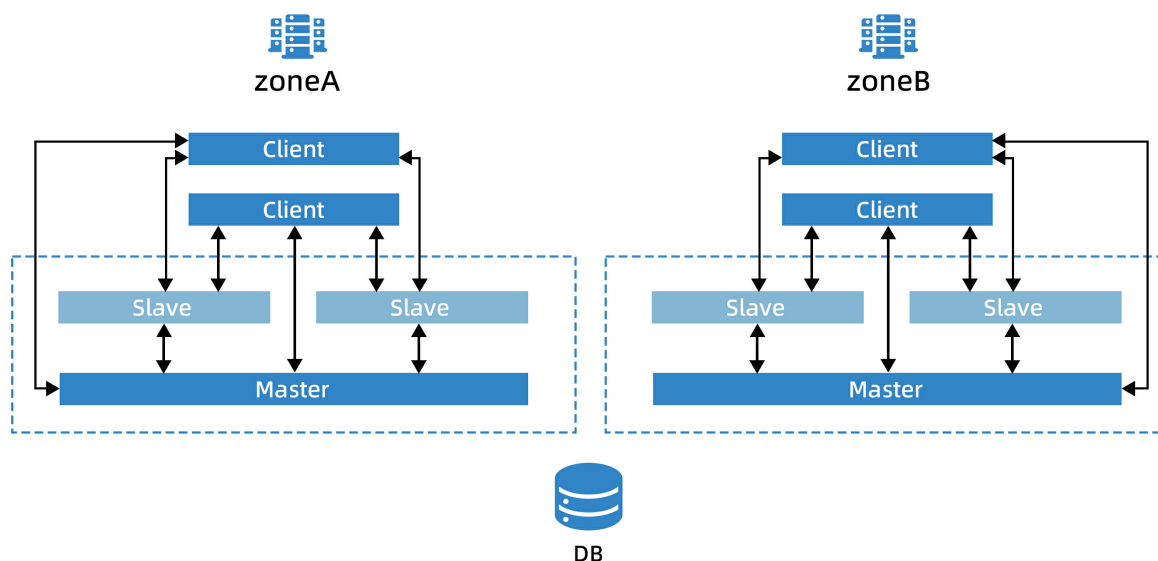
任务调度还需要有以下外部依赖：

**DB**：负责存储任务元数据、执行记录等信息。并作为任务调度集群注册表。

任务调度的过程如下：

1. 首先通过 console 添加任务，向调度器模块添加任务，并安排调度。
2. 当任务满足执行条件时，任务调度器则会触发任务调度，根据任务类型获取对应的调度执行器执行。
3. 调度执行器会通过事件方式创建触发记录、更新触发记录状态。
4. 调度器通过通讯模块通知客户端执行，RPC 采用 callback 模式下，客户端会将执行结果告诉服务端。服务端再根据事件模式更新任务执行状态。
5. 监听器监听漏触发和超时的执行记录，分别根据漏触发策略和超时策略进行处理。

### 分布式部署



如果任务单机部署，存在单点故障，一旦单机挂了，导致所有任务都不能执行。

考虑高可用性，目前大部分任务调度都具备分布式特性，在分布式场景下需要考虑以下几点：

- **任务分配**：每个任务只能分配给一台机器执行，避免出现并发调度情况。因此在任务调度分布式场景下都涉及到一个任务分配问题。
- **容灾机制**：当某个 slave 出现故障时，如何将分配给该 slave 上的任务分配给其他机器。

目前主流任务分布式调度系统，通过以下方式完成分布式模式：

- xxj-job 分布式模式依赖 quartz 的分布式管理，通过 quartz 来管理任务的分配。
- Elastic-Job 采用 ZK 来选主，通过 master 分配任务。

与其他框架不同，antscheduler 的选举模式依赖数据库，通过选举得到的 master 负责整个集群的任务分配和容灾。

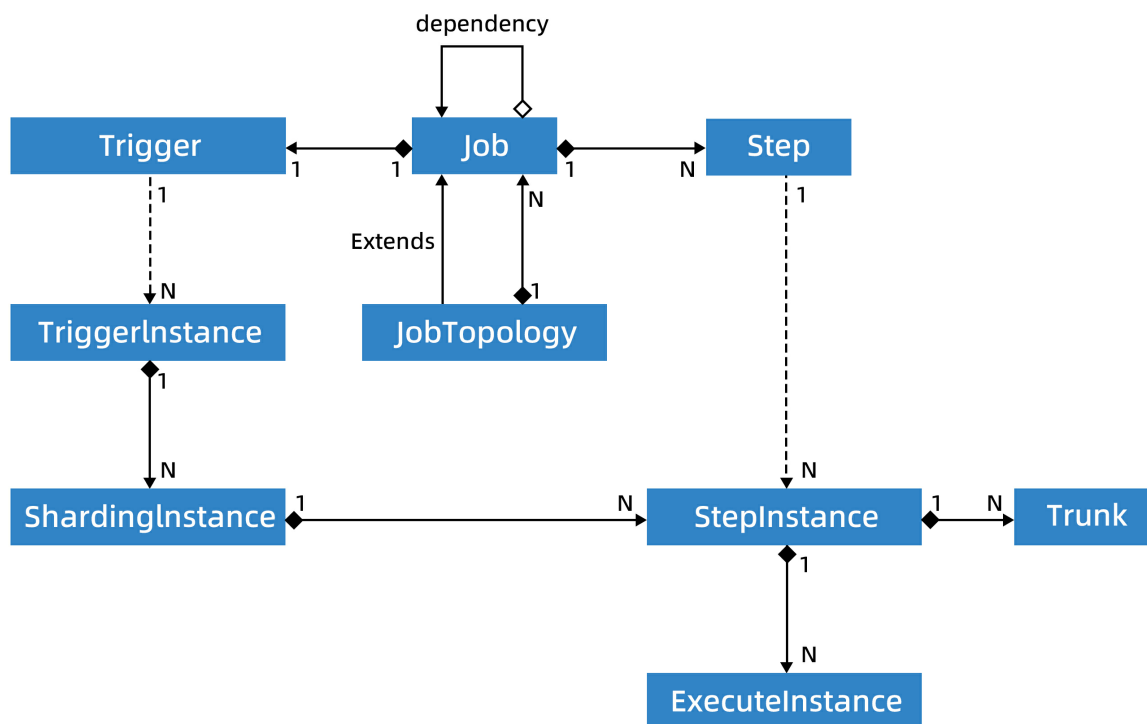
任务调度是一个分布式系统，避免单机承载过多任务，出现单点故障。通过分布式模式可以提高可用性、水平拓展能力。结合 LDC 模式，antscheduler 实现了 zone 级别的容灾能力。

Ant Scheduler 使用 quartz master/slave 架构，由 master hash 算法将任务分配到各个机器上，并且定期巡检 slave 状态异常时会把这台机器上的任务分配给其他机器来处理，这样就保证了当一台机器宕机时，受到影响的任务数量有限，达到高可用的目的。

## 核心模型



Ant Scheduler 的核心对象模型如下图所示：



## 4. 性能指标

不同业务场景，不同机器配置，性能不一样。性能数据可以参考以下压测报告。

### 背景

性能测试背景，针对哪些场景进行压测。

性能测试类型：基准测试、稳定性测试、容量测试、全链路压测

### 测试对象

AntScheduler server 端性能，包括：

- 单台 server 每秒能支持的简单事件任务（包括oneway/callback）最大的并发数，及性能瓶颈。
- 单台 server 定时任务的最大并发数。
- 简单任务最大分片支持数。
- 拓扑任务分片网关最大支持层级与分片数，拓扑任务最大节点数。
- 集群任务 chunk 最大拆分数量。
- 数据访问层的瓶颈，如执行记录数据多时出现大量慢 sql、数据库连接建立失败。
- 服务端内存缓存任务数量的瓶颈，单台任务量多大的需要进行扩容。

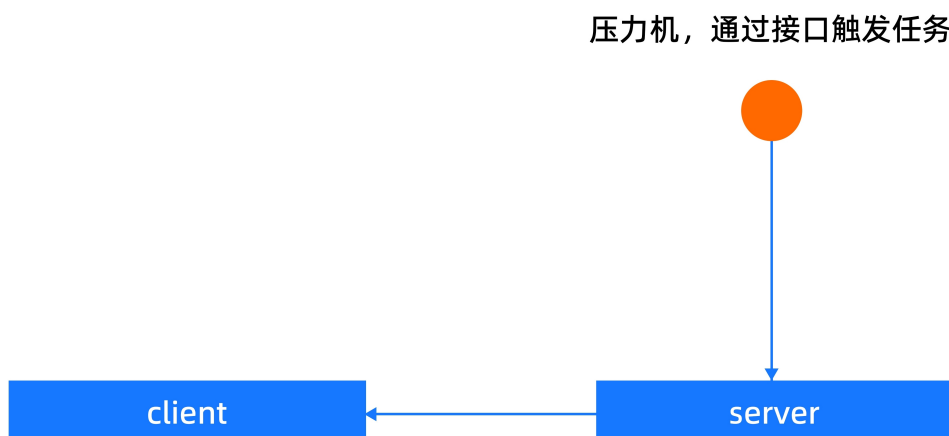
### 压测环境

机器配置如下：

- 施压机器：2C/4G \* N
- 被压机器：2C/4G, 1台

### 测试方案

1. 压测脚本中通过 rest 接口访问 server 触发指定的简单任务。



2. 在数据库添加多条定时任务，触发时间点相同，触发间隔都为 1s。
3. 前端限制分片数最大 100，需要在数据库修改任务的分片数为一个很大的值进行性能测试。

4. 构建一个分片网关层级比较深（3 层以上），调整网关分片数，确定一个最大阈值。
5. 创建一个 chunk 数量很大（>10000 个）的集群任务。

## 测试结果

### 压测场景一

通过 `/openapi/ts/job/trigger` 接口触发创建的简单 callback 任务，有以下问题：

- 单机 qps180 左右。压力机资源两台，antscheduler server 和 console 的系统 CPU、内存都还有空闲，GC 也不频繁，Fullgc 基本没触发。
- 响应时间太久，2 秒多。整个调用链路过长需要开发同学给链路中的所有调用加上耗时再进一步分析耗时问题。
- 压测过程中出现不少触发任务失败。由于触发同一个任务，触发频率太快（毫秒级），会造成任务触发实例创建失败，从而报错。

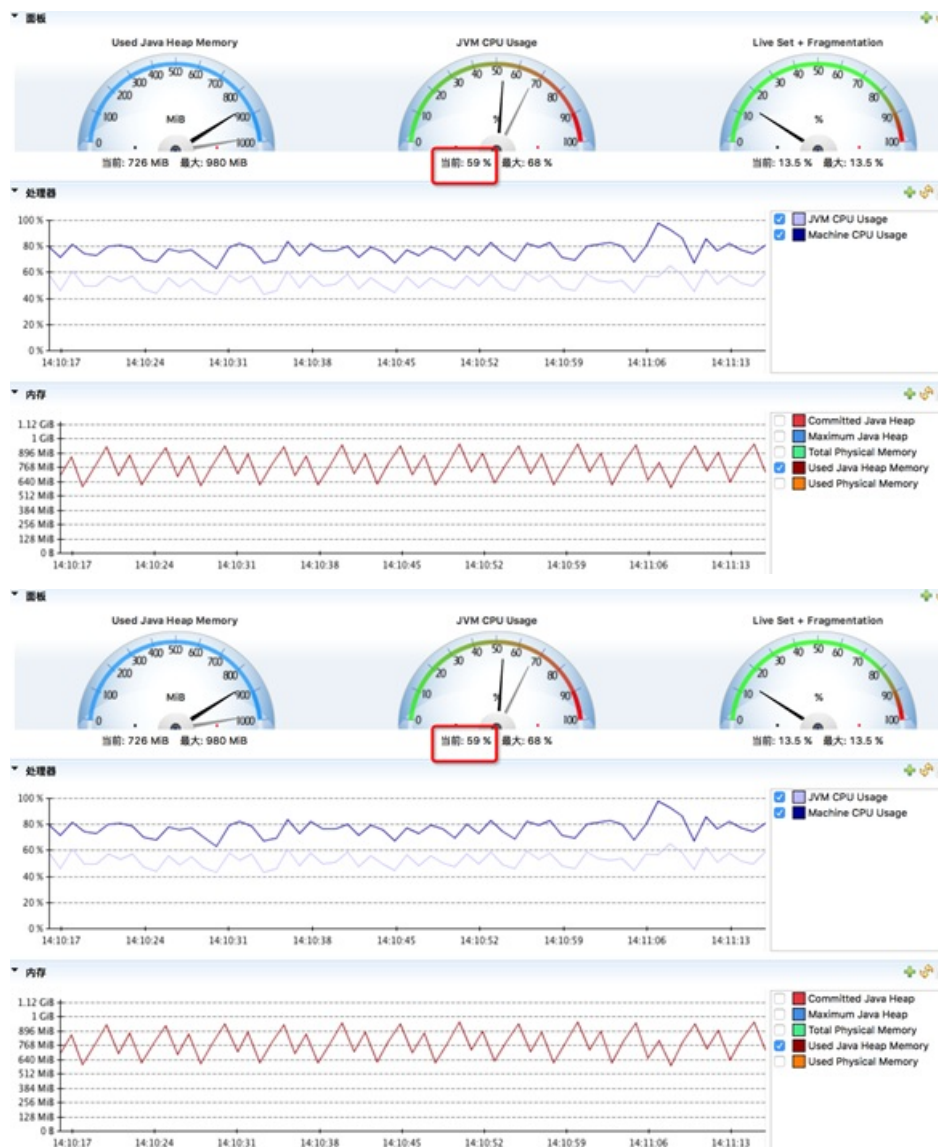
### 压测场景二

服务端任务下发线程池队列大小为 6000，所以总分片数超过该数量就会出现失败，考虑到不影响其他任务，建议单个任务最大分片数不超过 500（目前产品控制台创建任务只支持最大分片数为 100）。

针对分片为 500 的简单任务，客户端 job handler 线程池规模：core=10，maximumPoolSize=20，阻塞队列大小为 256，服务端创建对应 RPC 任务，分片数调为 500，服务端 JVM CPU 占用率达到 45% 以上，机器 CPU 使用率达 80% 左右。

以下数据为每秒触发一次分片为 500 的简单 RPC 任务相关性能数据：

total-cpu-usage							-dsk/total-		-net/total-		-paging-		-system-	
usr	sys	idl	wai	hiq	sig		read	writ	recv	send	in	out	int	csw
50	29	21					7136		4474	1863			87	20
56	27	17					11		4266	1831			81	20
45	28	27					8060		5164	1850			77	21
56	25	19					9288		4331	1880			75	21
43	26	31					6692		4357	1853			72	20
46	24	30					12		4466	1879			75	21
54	28	19					7040		4295	1855			77	20



### 压测场景三

集群任务主要有两个阶段：拆分和执行，拆分阶段服务端性能主要受 chunk 数量大小影响，而执行阶段服务端性能主要受执行 chunk 总并发数的影响。

集群任务配置：单台机器最大处理速率200，单台并行执行索引块数量上限 100，单台客户端。

- 集群任务 chunk 最大拆分数量：200\*100。



- 集群任务 chunk 最大拆分数量：200\*150。



- 集群任务 chunk 最大拆分数量：200\*200。

chunk 数据太多，前端页面渲染很卡。



## 5. 功能原理

### 5.1. 简单任务

简单任务是最基本的任务类型，适用于业务逻辑简单的场景。单个简单任务对应一个 handler，支持任务分片及分步：

- 当需要并发执行并且不关注资源利用率时，可以使用简单任务的分片的功能。
- 当需要将一个任务分成若干步骤按顺序执行时，可以使用简单任务的分步功能。

### 5.2. 集群任务

#### 处理阶段

集群任务将批处理分为两个大的阶段：拆分阶段和执行阶段，处理的核心数据模型为 Chunk（索引块）。

#### 拆分阶段

##### Chunk 拆分

拆分阶段类似于三层分发框架的 Split，不同的是可以多次拆分，适用于复杂场景。框架将拆分的数据封装为对象，上报到服务端并存储到数据库中。目前框架提供两种 IChunkData 类型：

- ShardingChunkData：分片 chunk，该 chunk 数据简单，只包含一个分片规则。
- RangeChunkData：范围 chunk，该 chunk 保存的是索引范围，需要填充起始索引，终止索引和分片规则。

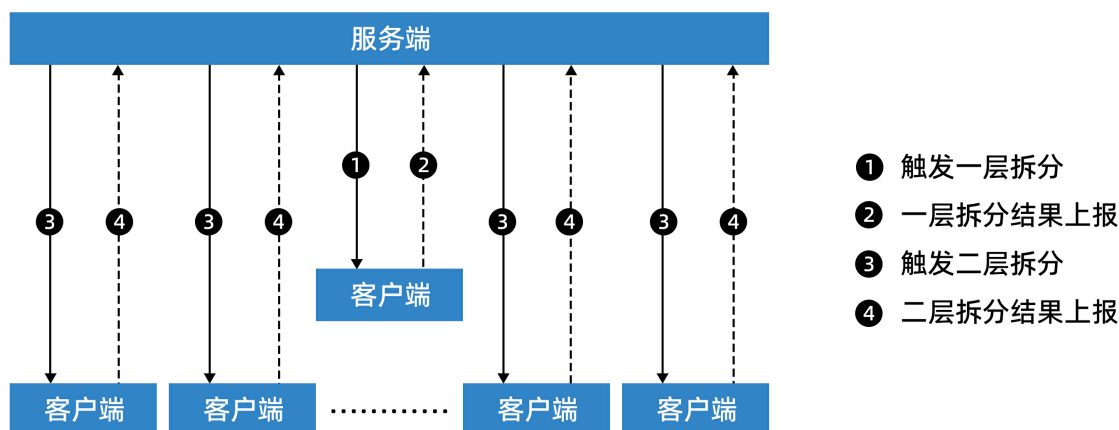
IChunkData 支持存储一些自定义属性到 extensionMap，这些数据会一起存储到数据库中，大小限制为 1024 个字符。

##### Chunk 上报

chunk 数据的长度为 10\*1000 个字符，超出长度后会休眠 3s 后再继续上报。若客户端未上报完信息时发生宕机，服务端感知到客户端的长连接断开，会再选择一台客户端重新进行拆分。

##### Chunk 入库

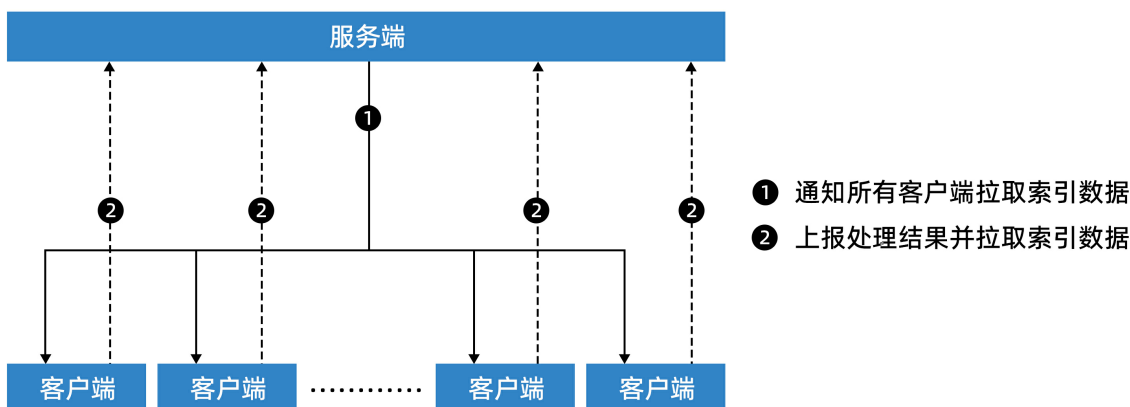
服务端在收到上报的 chunk 信息后，一次性把所有数据入库会给数据库带来较大压力，此处采取两阶段入库方式，即第一阶段先把 chunk 数据的信息入库，如分片规则，所属任务等，入库成功后会把 chunk 信息放入到缓冲队列中，然后依次批量更新 chunk 数据达到完整状态。若服务端未把 chunk 完整信息入库时宕机，会自动分配一台服务端重新触发拆分动作。



## 执行阶段

### 拉模式

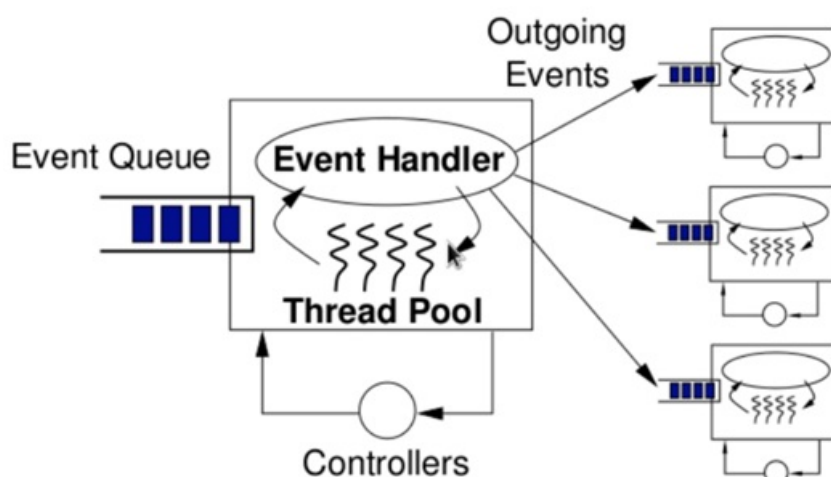
执行阶段不同于三层拆分框架，当数据拆分完毕后，服务端会通知客户端集群的所有机器来服务端拉取 chunk 信息，当拉取到的 chunk 处理完后，客户端会将结果上报到服务端，并且继续拉取新的 chunk。



chunk 的执行分为三个子阶段：Read，Process 和 Write，类似于 SpringBatch，执行逻辑非常契合 SEDA（阶段性事件驱动）架构思想，因此执行阶段采用 SEDA 架构进行了实现。

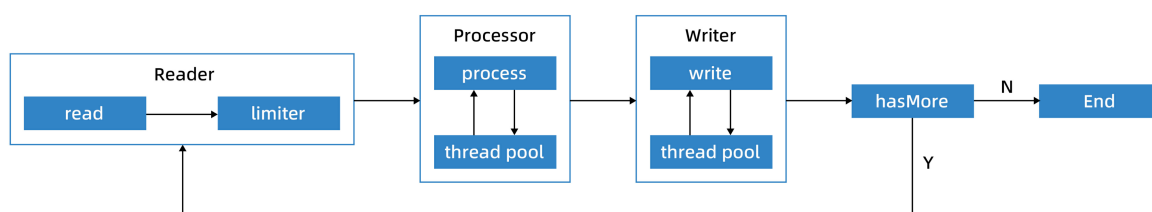
### SEDA 架构

SEDA 模型将一个任务分成了多个阶段，每个阶段拥有一个独立的事件缓冲队列和线程池，各个阶段之间通过事件进行通信。SEDA整合了多线程的服务器模型和事件驱动的服务器模型的优势，可以高效地管理和控制服务器资源，良好地适应高并发环境。

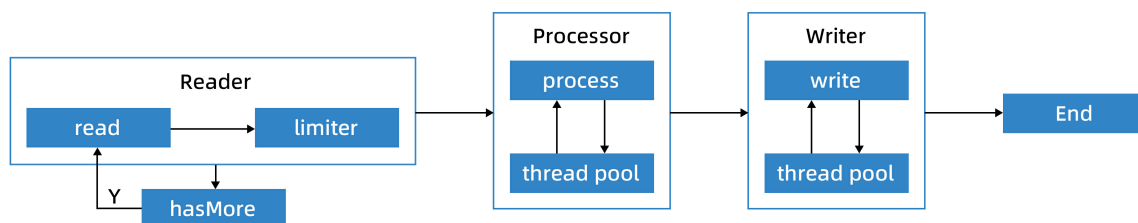


## 执行模式

同步模式：同步模式下，read write 完成之后，再开始下一次 read 操作。



异步模式：异步模式下，read 的数据全部放置到队列后即可开始下一次 read 操作，不需要等待write的执行结果。对于读操作比较耗时的场景，可以使用异步模式提高处理性能。



## 5.3. 拓扑任务

### 网关介绍

#### 开始与结束节点

所有任务拓扑有且仅有一个开始节点和一个结束节点。任务执行流程从开始节点起始，执行到结束节点终止。二者成对出现。

#### 条件网关

条件网关，即排它网关，也叫异或（XOR）网关，用来在流程中实现决策，成对出现。一对排它网关包括一个判断条件和两个执行分支，分别对应 true 和 false 两个判断结果。判断结果为 true 的分支将被执行。



## 并行网关

并行网关用来在流程中实现并发，成对出现。一对并行网关间的所有分支被同时执行，不进行条件判断。并行网关的一个分支执行完毕后，需要等待其他分支全部执行完流程才会走到下一个节点。

## 分片网关

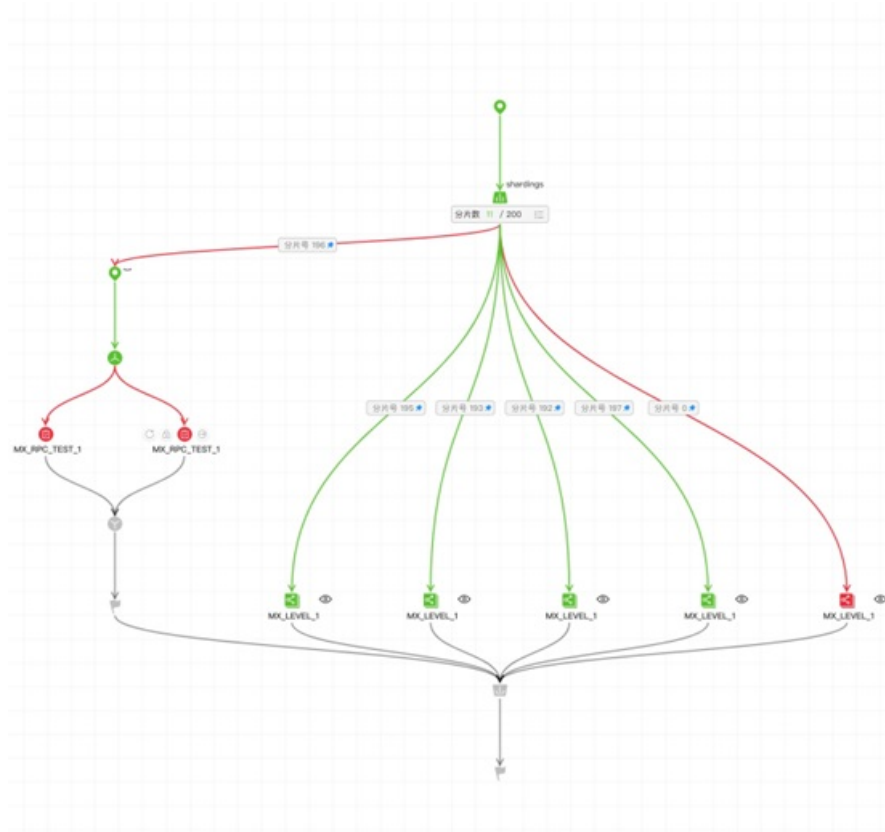
分片网关根据设定的分片维度对任务数据进行拆分，动态生成需要执行的分片数，成对出现。在拓扑图中使用分片网关时，需要在分片网关的开始节点配置分片维度。拓扑任务执行过程中，会根据拓扑任务执行上下文获取对应的分片配置，动态生成需要执行的分片。

## 子流程

支持拓扑任务嵌套拓扑任务支持多层嵌套，子流程之间参数隔离。

## 可视化操作

页面可视化操作，实时查询拓扑执行状态，并对执行过程进行人工干预操作。



## 版本管理

拓扑图定义支持多版本控制，可以基于当前发布的版本创建新版本。

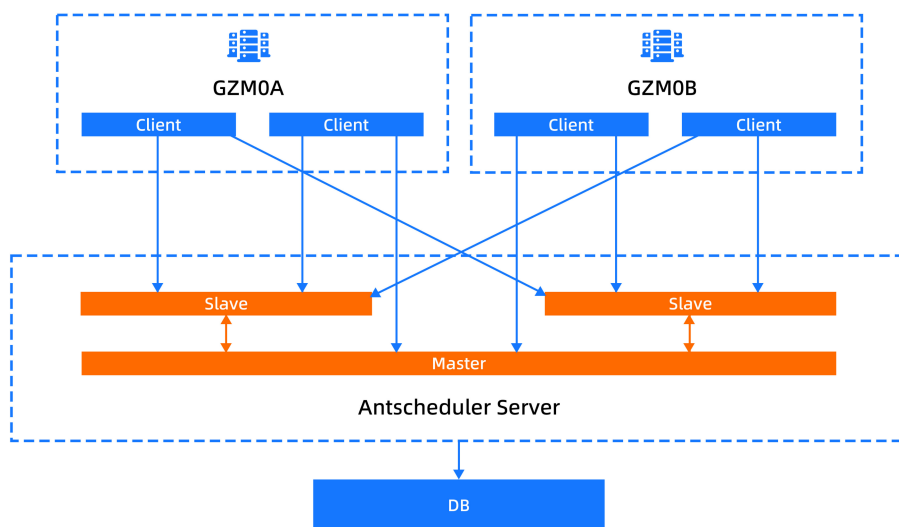
版本管理						
版本号	状态	基准版本	备注	发布时间	操作人	操作
V.2	已发布	V.1	- 父	2019-10-25 14:56:10	潘兴	查看
V.1	已归档	-	- 父	2019-10-25 14:32:10	潘兴	查看 发布 删除

## 6. 单元化能力

### 单元化架构

在单元化架构下，任务调度可以识别到任务客户端所在的单元信息，指定单元进行任务触发。具体架构实现流程如下：

1. 客户端连接所有的 server，注册连接时携带单元信息。
2. 服务端任务触发时，会根据启动的单元信息找到对应的连接，触发任务。



## 7.附录：基础术语

### 简单任务（Simple task）

最基本的任务类型，适用于业务逻辑简单的场景。单个简单任务对应一个 handler，支持任务分步。

### 集群任务（Cluster task）

支持在单个执行步骤中把数据拆分成多个数据分片（data chunk），把一个任务的数据分散到不同的机器上运行。适用于数据量很大的业务场景。调度过程分为拆分阶段和执行阶段。

### 任务拓扑（Topology）

由许多通过事件触发的任务的集合，描述了任务依赖关系。子任务可以并发执行或根据条件执行。

### 任务编排（Topology management）

将多个子任务根据业务逻辑编辑成拓扑任务的过程。

### 任务步骤（Step）

一个简单任务可以拆分成多个执行步骤，由不同的执行器（handler）按顺序执行。前置步骤成功完成后才开始执行下一个步骤。

### 数据分片（Data chunk）

当数据量很大时，任务调度系统支持在单个执行步骤中把数据拆分成多个分片，把一个任务的数据分散到不同的机器上运行。具体调度方式由调度中心根据负载进度进行调整。

### Cron 表达式（Cron expression）

固定格式的字符串，用来指定一个时间点或者一个重复触发的时间间隔。