# 入门

## 基本概念

任务{系统任务，工人任务}

工作流程{任务，子工作流}

工作流定义：基于JSON DSL定义

任务定义：输入，输出，超时，重试

系统任务：由conductor管理实现

工人任务：使用自己的语言程序实现通过rest/grpc实现与conductor服务器通信，轮询

## 客户端

Conductor 提供了以下 java 客户端来与各种 API 进行交互

| **客户** | **用法** |
| --- | --- |
| 元数据客户端 | 注册/更新工作流和任务定义 |
| 工作流客户端 | 启动新的工作流/获取工作流的执行状态 |
| 任务客户端 | 轮询任务/执行后更新任务结果/获取任务状态 |

TaskRunnerConfigurer 可用于注册工作线程并初始化轮询循环。  
管理任务工作线程池和服务器通信（轮询和任务更新）。

使用[Builder](https://github.com/Netflix/conductor/blob/master/client/src/main/java/com/netflix/conductor/client/automator/TaskRunnerConfigurer.java" \l "L62)创建 TaskRunnerConfigurer 的实例。构建器接受以下参数：

使用以下内容初始化构建器：TaskClient | 用于与Conductor 服务器通信的TaskClient | | 工人 | 将用于轮询工作和任务执行的工作线程。

| **参数** | **描述** | **默认** |
| --- | --- | --- |
| withEurekaClient | EurekaClient 用于识别服务器是否处于发现状态。当服务器退出发现时，轮询停止。如果传递 null，则不进行发现检查。 | 平台提供 |
| withThreadCount | 分配给工作人员的线程数。应该至少是 taskWorkers 的大小，以避免在繁忙的系统中饿死。 | 注册工人人数 |
| withSleepWhenRetry | 在重试操作之前，当任务更新调用失败时线程应该休眠的时间（以毫秒为单位）。 | 500 |
| withUpdateRetryCount | 更新状态调用失败时更新任务状态的尝试次数。 | 3 |
| withWorkerNamePrefix | 将用于所有工作人员的字符串前缀。 | 工作流-工作者- |

## 启动工作流

使用已注册的定义启动 Workflow 执行时，Workflow 接受以下参数：

| **field** | **description** | **Notes** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| name | 工作流的名称。必须在开始工作流程之前向 Conductor 注册 |  |
| version | 工作流版本 | 默认为最新的可用版本 |
| input | 具有键值参数的 JSON 对象，可供下游任务使用 | 有关详细信息，请参阅[接线输入和输出](https://github.com/weiwensi/conductor/blob/main/docs/configuration/workflowdef/" \l "wiring-inputs-and-outputs) |
| correlationId | 关联多个工作流执行的唯一 ID | 选修的 |
| taskToDomain | 有关详细信息，请参阅[任务域](https://github.com/weiwensi/conductor/blob/main/docs/configuration/taskdomains/" \l "task-domains)。 | 选修的 |
| workflowDef | 提供临时工作流定义以运行，无需注册。请参阅下面的动态工作流。 | 选修的 |
| externalInputPayloadStoragePath | 这由 Java 客户端负责。有关更多信息，请参阅[外部有效负载存储](https://github.com/weiwensi/conductor/blob/main/docs/externalpayloadstorage)。 | 选修的 |
| priority | 此工作流执行中任务的优先级。可能的值介于 0 - 99 之间。 | 选修的 |

{

"name": "encode\_and\_deploy",

"version": 1,

"correlationId": "my\_unique\_correlation\_id",

"input": {

"param1": "value1",

"param2": "value2"

}

}

## **动态工作流**

如果需要运行一次性工作流，并且在 Conductor Server 中注册任务和工作流定义没有意义，因为每次执行都会动态更改，则可以使用动态工作流执行。

这使您能够向workflowDef参数中的启动工作流请求提供嵌入了所需任务定义的工作流定义，从而避免在执行前注册蓝图。

****例子：****

使用有效负载发送POST请求，/workflow例如：

{

"name": "my\_adhoc\_unregistered\_workflow",

"workflowDef": {

"ownerApp": "my\_owner\_app",

"ownerEmail": "my\_owner\_email@test.com",

"createdBy": "my\_username",

"name": "my\_adhoc\_unregistered\_workflow",

"description": "Test Workflow setup",

"version": 1,

"tasks": [

{

"name": "fetch\_data",

"type": "HTTP",

"taskReferenceName": "fetch\_data",

"inputParameters": {

"http\_request": {

"connectionTimeOut": "3600",

"readTimeOut": "3600",

"uri": "${workflow.input.uri}",

"method": "GET",

"accept": "application/json",

"content-Type": "application/json",

"headers": {

}

}

},

"taskDefinition": {

"name": "fetch\_data",

"retryCount": 0,

"timeoutSeconds": 3600,

"timeoutPolicy": "TIME\_OUT\_WF",

"retryLogic": "FIXED",

"retryDelaySeconds": 0,

"responseTimeoutSeconds": 3000

}

}

],

"outputParameters": {

}

},

"input": {

"uri": "http://www.google.com"

}

}

# 配置

## 任务定义

Conductor 维护工人任务的注册表。任务必须在工作流中使用之前进行注册。

接口：[/metadata/taskdefs](http://localhost:8080/index.html" \l "!/Metadata_Management/registerTaskDef)，该接口是批量添加，文档中只给了单个任务，并且任务参数部分需要调整，下面为完整demo参数

[

{

"name": "encode",

"retryCount": 3,

"timeoutSeconds": 1200,

"pollTimeoutSeconds": 1100,

"inputKeys": [

"sourceRequestId",

"qcElementType"

],

"outputKeys": [

"state",

"skipped",

"result"

],

"timeoutPolicy": "TIME\_OUT\_WF",

"retryLogic": "FIXED",

"retryDelaySeconds": 600,

"responseTimeoutSeconds": 1100,

"concurrentExecLimit": 100,

"rateLimitFrequencyInSeconds": 60,

"rateLimitPerFrequency": 50,

"ownerEmail": "encode\_admin@test.com"

},{

"name": "deploy",

"retryCount": 3,

"timeoutSeconds": 1200,

"pollTimeoutSeconds": 1100,

"inputKeys": [

"sourceRequestId",

"qcElementType"

],

"outputKeys": [

"state",

"skipped",

"result"

],

"timeoutPolicy": "TIME\_OUT\_WF",

"retryLogic": "FIXED",

"retryDelaySeconds": 600,

"responseTimeoutSeconds": 1100,

"concurrentExecLimit": 100,

"rateLimitFrequencyInSeconds": 60,

"rateLimitPerFrequency": 50,

"ownerEmail": "encode\_admin@test.com"

}

]

| **field** | **description** | **Notes** |
| --- | --- | --- |
| name | 任务类型。与其功能产生共鸣的任务的唯一名称。 | Unique |
| description | 任务描述 | 选修的 |
| retryCount | 任务被标记为失败时的重试次数 | 默认为 3 |
| retryLogic | 重试机制 | 请参阅下面的可能值 |
| retryDelaySeconds | 重试前等待的时间 | 默认为 60 秒 |
| timeoutPolicy | 任务的超时策略 | 请参阅下面的可能值 |
| timeoutSeconds | 以秒为单位的时间，在此之后任务在第一次TIMED\_OUT转换到IN\_PROGRESS状态后被标记为未完成 | 如果设置为 0，则没有超时 |
| pollTimeoutSeconds | 以秒为单位的时间，在此之后任务被标记为TIMED\_OUT没有被工作人员轮询 | 如果设置为 0，则没有超时 |
| responseTimeoutSeconds | 必须大于 0 且小于 timeoutSeconds。如果在此时间之后没有更新状态（心跳机制），则重新安排任务。当工作人员轮询任务但由于错误/网络故障而无法完成时很有用。 | 默认为 3600 |
| inputKeys | 任务预期输入的键数组。用于记录任务的输入。请参阅[使用 inputKeys 和 outputKeys](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/taskdef.md" \l "using-inputkeys-and-outputkeys)。 | 选修的 |
| outputKeys | 任务预期输出的键数组。用于记录任务的输出 | 选修的 |
| inputTemplate | 请参阅下面的[使用 inputTemplate](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/taskdef.md" \l "using-inputtemplate)。 | 选修的 |
| concurrentExecLimit | 在任何给定时间可以执行的任务数。 | 选修的 |
| rateLimitFrequencyInSeconds, rateLimitPerFrequency | 请参阅下面的[任务速率限制](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/taskdef.md" \l "task-rate-limits)。 | 选修的 |
| ownerEmail | 包含任务所有者电子邮件的字符串。 | 强制除非禁用 |
| isolationGroupId | 请参阅[隔离组](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/isolationgroups.md" \l "Isolation-Group-Id)。 | 选修的 |
| executionNameSpace | 请参阅[执行名称空间](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/isolationgroups.md" \l "Execution-Name-Space)。 | 选修的 |

### **重试逻辑**

* FIXED ：在任务结束后重新安排任务 retryDelaySeconds
* EXPONENTIAL\_BACKOFF ：在之后重新安排 retryDelaySeconds \* attemptNumber

### **超时政策**

* RETRY : 再次重试任务
* TIME\_OUT\_WF ：工作流被标记为 TIMED\_OUT 并终止
* ALERT\_ONLY ：注册一个计数器（task\_timeout）

### **任务并发执行限制**

* concurrentExecLimit在任何时候限制同时执行的任务数量。  
  ****示例：****  
  如果您有 1000 个任务在队列中等待执行，并且有 1000 个工作人员轮询此队列以查找任务，但如果您设置concurrentExecLimit为 10，则只会将 10 个任务分配给工作人员（这会导致饥饿）。如果任何一个工作线程完成执行，一个或多个新任务将从队列中删除，同时仍然保持当前执行计数为 10。

### **任务速率限制**

* rateLimitFrequencyInSeconds并且rateLimitPerFrequency应该一起使用。
* rateLimitFrequencyInSeconds设置“频率窗口”，即duration要在events per duration. 例如：1 秒、5 秒、60 秒、300 秒等。
* rateLimitPerFrequency定义了每个给定的“频率窗口”可以提供给工作人员的任务数量。  
  ****示例：****  
  让我们设置rateLimitFrequencyInSeconds = 5, 和rateLimitPerFrequency = 12。这意味着，我们的频率窗口的持续时间为 5 秒，对于每个频率窗口，Conductor 只会给工人分配 12 个任务。因此，在给定的一分钟内，Conductor 只会向工作人员分配 12\*(60/5) = 144 个任务，而不管轮询任务的工作人员数量如何。  
  请注意，与 不同的是concurrentExecLimit，速率限制不考虑已经在进行中/已完成的任务。即使之前的所有任务都在 1 秒内执行，或者需要几天时间，新任务仍然以配置的频率分配给工作人员，在上面的例子中每分钟 144 个任务。  
  注意：速率限制仅支持 Redis-persistence 模块，不适用于其他持久层。

### **使用 inputKeys 和 outputKeys**

* inputKeys并且outputKeys可以被认为是任务的参数和返回值。
* 将任务定义视为由接口表示： (value1, value2 .. valueN) someTaskDefinition(key1, key2 .. keyN);
* 但是，这些参数目前并未严格执行。双方inputKeys并outputKeys充当任务重新使用一个文档。工作流中的任务不需要定义任务定义中的所有键。
* 将来，这可以扩展为所有任务实现都必须遵守的严格模板，就像编程语言中的接口一样。

### **使用输入模板**

* inputTemplate 允许定义默认值，这些值可以被 Workflow 中提供的值覆盖。
* 例如：在您的任务定义中，您可以将 inputTemplate 定义为：

“inputTemplate”：{

“url”：“ https://some\_url:7004 ”

}

* 现在，在您的工作流定义中，使用上述任务时，您可以使用默认值url或覆盖任务的inputParameters.

"inputParameters": {

"url": "${workflow.input.some\_new\_url}"

}

## 工作流定义

工作流是使用基于 JSON 的 DSL 定义的

Demo:

{

"name": "encode\_and\_deploy",

"description": "Encodes a file and deploys to CDN",

"version": 1,

"tasks": [

{

"name": "encode",

"taskReferenceName": "encode",

"type": "SIMPLE",

"inputParameters": {

"fileLocation": "${workflow.input.fileLocation}"

}

},

{

"name": "deploy",

"taskReferenceName": "d1",

"type": "SIMPLE",

"inputParameters": {

"fileLocation": "${encode.output.encodeLocation}"

}

}

],

"outputParameters": {

"cdn\_url": "${d1.output.location}"

},

"failureWorkflow": "cleanup\_encode\_resources",

"restartable": true,

"workflowStatusListenerEnabled": true,

"schemaVersion": 2,

"ownerEmail": "encode\_admin@test.com"

}

| name | **描述** | **笔记** |
| --- | --- | --- |
| Name | 工作流名称 |  |
| description | 工作流程描述 | 可选 |
| version | 用于标识架构版本的数字字段。使用递增数字 | 启动工作流执行时，如果未指定，则使用版本最高的定义 |
| tasks | 如下所述的一系列任务定义。 |  |
| inputParameters | 输入参数列表。用于记录工作流所需的输入 | 可选 |
| outputParameters | 用于生成工作流输出的 JSON 模板 | 如果未指定，则输出定义为上次执行任务的输出 |
| failureWorkflow | 细绳; 要在当前工作流失败时运行的工作流。用于清理或发布失败的操作。 | 可选 |
| schemaVersion | 当前导体架构版本。schemaVersion 1 已停止使用。 | 必须是 2 |
| restartable | 允许工作流重新启动的布尔标志 | 默认为true |
| workflowStatusListenerEnabled | 如果为 true，则每个终止或完成的工作流都会发送通知。见[下文](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/workflowdef.md" \l "workflow-notifications) | 可选（默认为 false） |
| ownerEmail | 包含工作流所有者电子邮件的字符串。 | 强制除非禁用 |
| timeoutSeconds | 以秒为单位的时间，在此之后工作流超时 | 如果设置为 0，则没有超时（默认为 0） |
| timeoutPolicy | 工作流的超时策略 | 请参阅下面的可能值（默认为 ALERT\_ONLY） |

### **超时政策**

* TIME\_OUT\_WF ：工作流被标记为 TIMED\_OUT 并终止。
* ALERT\_ONLY ：记录超时但工作流未终止。

### 工作流中的任务

tasks 工作流执行中的属性定义了要按该顺序执行的任务数组。

| field | 描述 | 笔记 |
| --- | --- | --- |
| name | 任务名称。必须在开始工作流程之前向 Conductor 注册为任务 |  |
| taskReferenceName | 别名用于引用工作流中的任务。在工作流中必须是唯一的。 |  |
| type | 任务类型。SIMPLE 用于远程工作者执行的任务，或系统任务类型之一 |  |
| description | 任务描述 | 选修的 |
| optional | 对或错。设置为 true 时 - 即使任务失败，工作流也会继续。任务的状态反映为COMPLETED\_WITH\_ERRORS | 默认为 false |
| inputParameters | 定义给任务的输入的 JSON 模板 | 有关详细信息，请参阅[接线输入和输出](https://github.com/weiwensi/conductor/blob/dev/docs/docs/configuration/workflowdef.md" \l "wiring-inputs-and-outputs) |
| domain | 有关详细信息，请参阅[任务域](https://github.com/weiwensi/conductor/blob/dev/conductor/configuration/taskdomains)。 | 选修的 |

除了这些参数之外，系统任务还有自己的参数.Checkout [System Tasks](https://github.com/weiwensi/conductor/blob/dev/conductor/configuration/systask) for more information.

## 事件处理器（****eventhandlers****）

### 事件处理程序

事件处理程序是注册的侦听器，当匹配事件发生时执行操作。支持的操作是：

1. 启动工作流
2. 任务失败
3. 完成一项任务

事件处理程序是通过/event/API配置的。

#### **结构：**

{

"name" : "描述唯一的名称

",

"event": "event\_type:event\_location",

"condition": "boolean condition",

"actions": ["参见下面的示例

"]

}

****Start A Workflow****

{

"action": "start\_workflow",

"start\_workflow": {

"name": "WORKFLOW\_NAME",

"version": "<optional\_param>",

"input": {

"param1": "${param1}"

}

}

}

****Complete Task****\*

{

"action": "complete\_task",

"complete\_task": {

"workflowId": "${workflowId}",

"taskRefName": "task\_1",

"output": {

"response": "${result}"

}

},

"expandInlineJSON": true

}

****Fail Task****\*

{

"action": "fail\_task",

"fail\_task": {

"workflowId": "${workflowId}",

"taskRefName": "task\_1",

"output": {

"response": "${result}"

}

},

"expandInlineJSON": true

}

Demo：

开始一个工作流

{

"active":true,

"name":"startOneFlow",

"event":"conductor:{com.netflix.conductor.core.events.queue.dyno.DynoEventQueueProvider}",

"condition":"1==1",

"actions":[

{

"action":"start\_workflow",

"start\_workflow":{

"name":"encode\_deploy222222",

"input":{

"param1":"${param1}"

}

}

}

]

}