Apache ShardingSphere ElasticJob document

Apache ShardingSphere

Jan 19, 2022

Contents

1 Introduction **2**
- 1.1 ElasticJob-Lite . . . 2
- 1.2 ElasticJob-Cloud . . . 3

2 Features **4**

3 Environment Required **5**
- 3.1 Java . . . 5
- 3.2 Maven . . . 5
- 3.3 ZooKeeper . . . 5
- 3.4 Mesos (ElasticJob-Cloud only) . . . 5

4 Quick Start **6**
- 4.1 ElasticJob-Lite . . . 6
  - 4.1.1 Import Maven Dependency . . . 6
  - 4.1.2 Develop Job . . . 6
  - 4.1.3 Configure Job . . . 7
  - 4.1.4 Schedule Job . . . 7
- 4.2 ElasticJob-Cloud . . . 7
  - 4.2.1 Import Maven Dependency . . . 7
  - 4.2.2 Develop Job Details . . . 8
  - 4.2.3 Develop Job Bootstrap . . . 8
  - 4.2.4 Pack Job . . . 8
  - 4.2.5 API Authentication . . . 8
  - 4.2.6 Publish Job . . . 9
  - 4.2.7 Schedule Job . . . 9

5 Concepts & Features **10**
- 5.1 Schedule Model . . . 10
  - 5.1.1 In-process scheduling . . . 10
  - 5.1.2 Process-level scheduling . . . 10
- 5.2 Elastic Schedule . . . 11

5.2.1 Sharding . . . 11
Sharding Item . . . 12
Customized sharding options . . . 12
5.2.2 Maximize the usage of resources . . . 12
5.2.3 High Availability . . . 13
5.2.4 ElasticJob-Lite Implementation Principle . . . 13
Elastic Distributed Implementation . . . 14
Registry Data Structure . . . 14
config node . . . 14
instances node . . . 14
sharding node . . . 15
servers node . . . 15
leader node . . . 15
5.3 Resource Assign . . . 16
5.3.1 Execution mode . . . 17
Transient execution . . . 17
Daemon execution . . . 17
5.3.2 Scheduler . . . 17
5.3.3 Job Application . . . 17
5.3.4 Job . . . 17
5.3.5 Resource . . . 17
5.4 Failover . . . 18
5.4.1 Concept . . . 18
5.4.2 Execution mechanism . . . 20
Notification execution . . . 20
Enquiry execution . . . 20
5.4.3 Scenarios: . . . 20
5.5 Misfire . . . 20
5.5.1 Concept . . . 21
5.5.2 Scenarios . . . 22
5.6 Job Open Ecosystem . . . 22
5.6.1 Job interface . . . 22
5.6.2 Actuator interface . . . 23

6 User Manual **24**
6.1 ElasticJob-Lite . . . 24
6.1.1 Introduction . . . 24
6.1.2 Comparison . . . 25
6.1.3 Usage . . . 25
Job API . . . 25
Job Listener . . . 47
Tracing . . . 51
Operation API . . . 56
6.1.4 Configuration . . . 60
Registry Center Configuration . . . 60

Job Configuration . . . 61
Job Listener Configuration . . . 63
Event Tracing Configuration . . . 63
Java API . . . 63
Spring Boot Starter . . . 65
Spring Namespace . . . 68
Built-in Strategy . . . 71
Job Properties . . . 76
6.1.5 Operation . . . 77
Deploy Guide . . . 77
Dump Job Information . . . 78
Execution Monitor . . . 79
Console . . . 79
6.2 ElasticJob-Cloud . . . 80
6.2.1 Introduction . . . 80
6.2.2 Comparison . . . 81
6.2.3 Usage . . . 81
Dev Guide . . . 81
Local Executor . . . 82
6.2.4 Configuration . . . 82
Authentication API . . . 82
Application API . . . 83
Job API . . . 85
6.2.5 Operation . . . 87
Deploy Guide . . . 87
High Available . . . 89
Console . . . 89

7 Dev Manual **91**
7.1 Job Sharding Strategy . . . 91
7.2 Thread Pool Strategy . . . 92
7.3 Error Handler . . . 92
7.4 Job Class Name Provider . . . 92
7.5 Roadmap . . . 93
7.5.1 Kernel . . . 93
7.5.2 ElasticJob-Lite . . . 93
7.5.3 ElasticJob-Cloud . . . 94

8 Downloads **96**
8.1 Latest Releases . . . 96
8.1.1 ElasticJob - Version: 3.0.1 (Release Date: Oct 11, 2021) . . . 96
8.1.2 ElasticJob-UI - Version: 3.0.1 (Release Date: Jan 19, 2022) . . . 96
8.2 All Releases . . . 96
8.3 Verify the Releases . . . 97

9 Powered By **98**

9.1 Register . . . 98
9.2 Who are using ElasticJob? . . . 98
9.2.1 E-commerce . . . 98
9.2.2 Financial Industry . . . 99
9.2.3 Digitalization and Cloud Services . . . 99
9.2.4 Transportation . . . 99
9.2.5 Logistics . . . 100
9.2.6 Real Estate . . . 100
9.2.7 E-education . . . 100
9.2.8 E-entertainment . . . 100
9.2.9 News . . . 100
9.2.10 Communication . . . 101
9.2.11 Internet of Things . . . 101
9.2.12 Software Development Services . . . 101
9.2.13 Health Care . . . 101
9.2.14 Retail . . . 102
9.2.15 AI . . . 102

10 FAQ **103**
10.1 Why do some compiling errors appear? . . . 103
10.2 Does ElasticJob support dynamically adding jobs? . . . 103
10.3 Why is the job configuration modified in the code or Spring XML file, but the registry center is not updated? . . . 104
10.4 What happens if the job can't communicate with the registry center? . . . 104
10.5 What are the usage restrictions of `ElasticJob-Lite`? . . . 104
10.6 What should you do if you suspect that `ElasticJob-Lite` has a problem in a distributed environment, but it cannot be reproduced and cannot be debugged in the online environment? . . . 105
10.7 What are the usage restrictions of `ElasticJob-Cloud`? . . . 105
10.8 When add a task in the `ElasticJob-Cloud`, why does it remain in the ready state, but doesn't start? . . . 105
10.9 Why can't the Console page display normally? . . . 105
10.10 Why is the job state shard to be adjusted in the Console? . . . 106
10.11 Why is there a task scheduling delay in the first startup? . . . 106
10.12 In Windows env, run ShardingSphere-ElasticJob-UI, could not find or load main class org.apache.shardingsphere.elasticjob.lite.ui.Bootstrap. Why? . . . 106
10.13 Unable to startup Cloud Scheduler. Continuously output "Elastic job: IP:PORT has leadership" . . . 106
10.14 Unable to obtain a suitable IP in the case of multiple network interfaces . . . 107

11 Blog **108**

ElasticJob is a distributed scheduling solution consisting of two separate projects, ElasticJob-Lite and ElasticJob-Cloud.

Through the functions of flexible scheduling, resource management and job management, it creates a distributed scheduling solution suitable for Internet scenarios, and provides a diversified job ecosystem through open architecture design. It uses a unified job API for each project. Developers only need code one time and can deploy at will.

ElasticJob became an Apache ShardingSphere Sub project on May 28 2020.

Welcome communicate with community via mail list.

1 Introduction

Using ElasticJob can make developers no longer worry about the non-functional requirements such as jobs scale out, so that they can focus more on business coding; At the same time, it can release operators too, so that they do not have to worry about jobs high availability and management, and can automatic operation by simply adding servers.

1.1 ElasticJob-Lite

A lightweight, decentralized solution that provides distributed task sharding services.

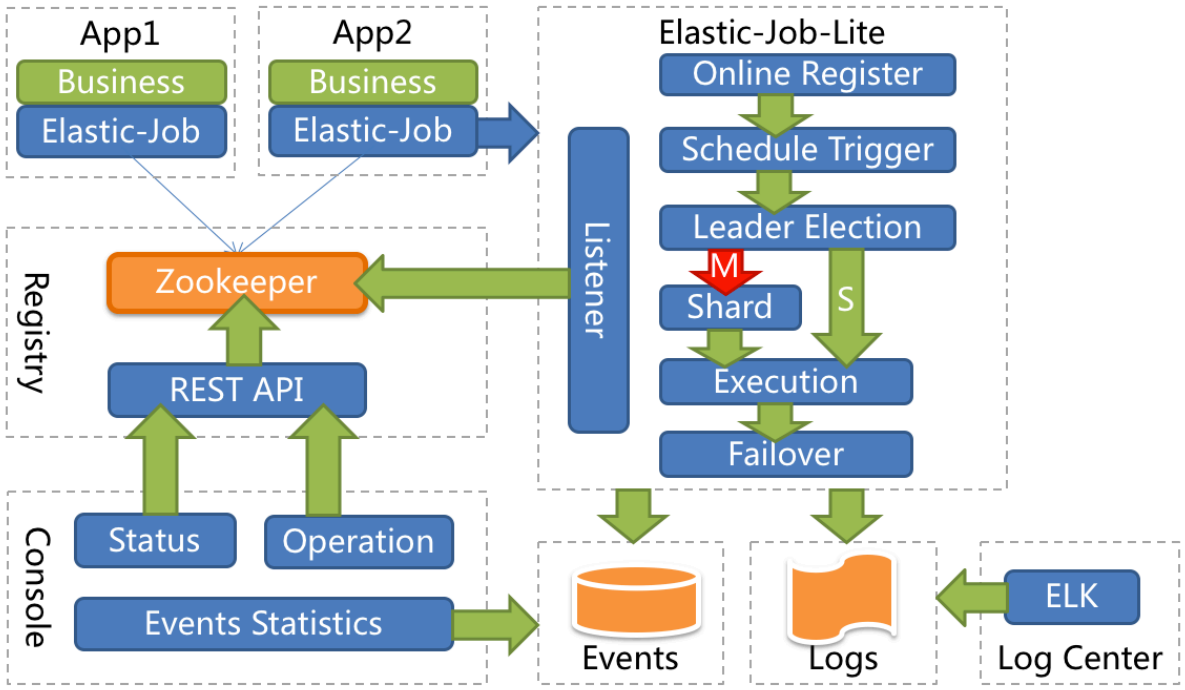

Figure1: ElasticJob-Lite Architecture

1.2 ElasticJob-Cloud

Uses Mesos to manage and isolate resources.

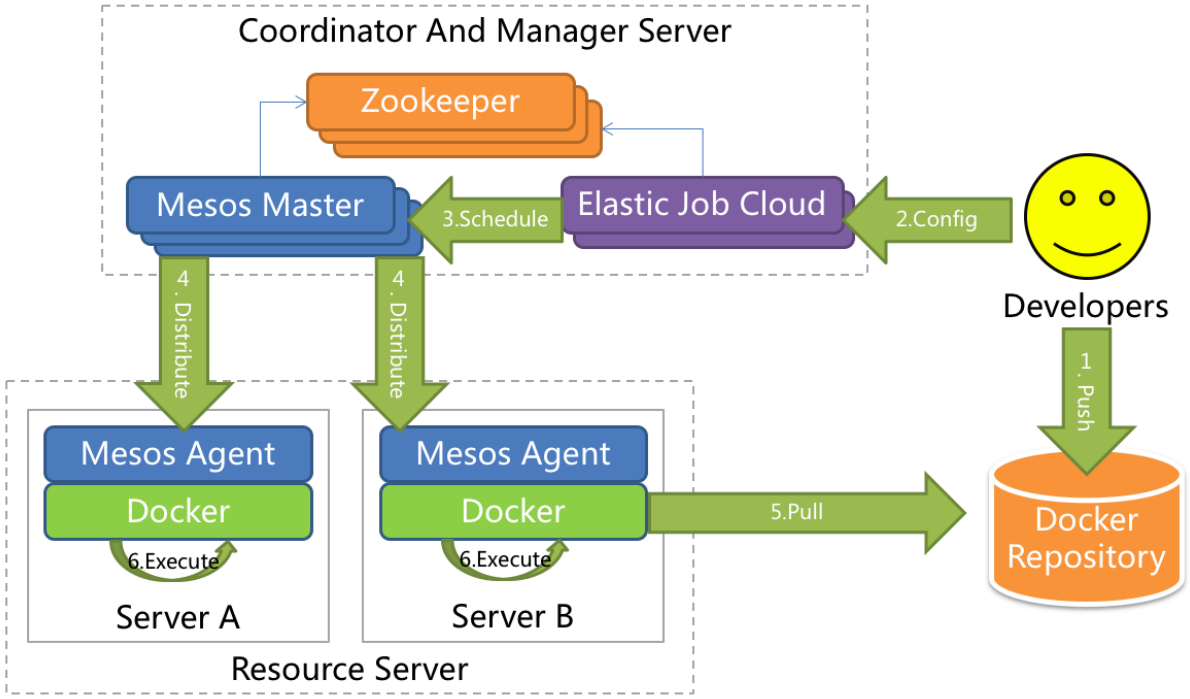

Figure2: ElasticJob-Cloud Architecture

	ElasticJob-Lite	*ElasticJob-Cloud*
Decentralization	Yes	No
Resource Assign	No	Yes
Job Execution	Daemon	Daemon + Transient
Deploy Dependency	ZooKeeper	ZooKeeper + Mesos

2 Features

- Elastic Schedule
  - Support job sharding and high availability in distributed system
  - Scale out for throughput and efficiency improvement
  - Job processing capacity is flexible and scalable with the allocation of resources
- Resource Assign
  - Execute job on suitable time and assigned resources
  - Aggregation same job to same job executor
  - Append resources to newly assigned jobs dynamically
- Job Governance
  - Failover
  - Misfired
  - Self diagnose and recover when distribute environment unstable
- Job Dependency (TODO)
  - DAG based job dependency
  - DAG based job item dependency
- Job Open Ecosystem
  - Unify job api for extension
  - Support rich job type lib, such as dataflow, script, HTTP, file, big data
  - Focus business SDK, can work with Spring IOC
- Admin Console
  - Job administration
  - Job event trace query
  - Registry center management

3 Environment Required

3.1 Java

Java 8 or above required.

3.2 Maven

Maven 3.5.0 or above required.

3.3 ZooKeeper

ZooKeeper 3.6.0 or above required. See details

3.4 Mesos (ElasticJob-Cloud only)

Mesos 1.1.0 or compatible version required. See details

4 Quick Start

In shortest time, this chapter provides users with a simplest quick start with ElasticJob.

4.1 ElasticJob-Lite

4.1.1 Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-lite-core</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

4.1.2 Develop Job

```
public class MyJob implements SimpleJob {

    @Override
    public void execute(ShardingContext context) {
        switch (context.getShardingItem()) {
            case 0:
                // do something by sharding item 0
                break;
            case 1:
                // do something by sharding item 1
                break;
            case 2:
                // do something by sharding item 2
                break;
            // case n: ...
        }
```

```
    }
}
```

4.1.3 Configure Job

```
JobConfiguration jobConfig = JobConfiguration.newBuilder("MyJob", 3).cron("0/5 * *
* * * ?").build();
```

4.1.4 Schedule Job

```
public class MyJobDemo {

    public static void main(String[] args) {
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createJobConfiguration()).schedule();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(new
ZookeeperConfiguration("zk_host:2181", "my-job"));
        regCenter.init();
        return regCenter;
    }

    private static JobConfiguration createJobConfiguration() {
        // create job configuration
        // ...
    }
}
```

4.2 ElasticJob-Cloud

4.2.1 Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-cloud-executor</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

4.2.2 Develop Job Details

```
public class MyJob implements SimpleJob {

    @Override
    public void execute(ShardingContext context) {
        switch (context.getShardingItem()) {
            case 0:
                // do something by sharding item 0
                break;
            case 1:
                // do something by sharding item 1
                break;
            case 2:
                // do something by sharding item 2
                break;
            // case n: ...
        }
    }
}
```

4.2.3 Develop Job Bootstrap

Define `main` method and call `JobBootstrap.execute()`, example as follows:

```
public class MyJobDemo {

    public static void main(final String[] args) {
        JobBootstrap.execute(new MyJob());
    }
}
```

4.2.4 Pack Job

```
tar -cvf my-job.tar.gz my-job
```

4.2.5 API Authentication

```
curl -H "Content-Type: application/json" -X POST http://elasticjob_cloud_host:8899/
api/login -d '{"username": "root", "password": "pwd"}'
```

Response body:

```
{"accessToken":"some_token"}
```

4.2.6 Publish Job

```
curl -l -H "Content-type: application/json" -H "accessToken: some_token" -X POST -d
'{"appName":"my_app","appURL":"http://app_host:8080/my-job.tar.gz","cpuCount":0.1,
"memoryMB":64.0,"bootstrapScript":"bin/start.sh","appCacheEnable":true,
"eventTraceSamplingCount":0}' http://elasticjob_cloud_host:8899/api/app
```

4.2.7 Schedule Job

```
curl -l -H "Content-type: application/json" -H "accessToken: some_token" -X POST -d
'{"jobName":"my_job","appName":"my_app","jobExecutionType":"TRANSIENT","cron":"0/5
* * * * ?","shardingTotalCount":3,"cpuCount":0.1,"memoryMB":64.0}' http://
elasticjob_cloud_host:8899/api/job/register
```

5 Concepts & Features

This chapter describes concepts and features about ElasticJob. Please refer to User manual for more details.

5.1 Schedule Model

Unlike most job platforms, ElasticJob's scheduling model is divided into in-process scheduling ElasticJob-Lite that supports thread-level scheduling, and ElasticJob-Cloud for process-level scheduling.

5.1.1 In-process scheduling

ElasticJob-Lite is a thread-level scheduling framework for in-process. Through it, Job can be transparently combined with business application systems. It can be easily used in conjunction with Java frameworks such as Spring and Dubbo. Spring DI (Dependency Injection) Beans can be freely used in Job, such as data source connection pool and Dubbo remote service, etc., which is more convenient for business development.

5.1.2 Process-level scheduling

ElasticJob-Cloud has two methods: in-process scheduling and process-level scheduling. Because ElasticJob-Cloud can control the resources of the job server, its job types can be divided into resident tasks and transient tasks. The resident task is similar to ElasticJob-Lite, which is an in-process scheduling; the transient task is completely different. It fully utilizes the peak-cutting and valley-filling capabilities of resource allocation, and is a process-level scheduling. Each task will start a new process.

5.2 Elastic Schedule

Elastic schedule is the most important feature in ElasticJob, which acts as a job processing system that enables the horizontal scaling of jobs by sharding, it’s also the origin of the project name “ElasticJob”.

5.2.1 Sharding

A concept in ElasticJob to split the job, enabling the job to be executed in distributed environment, where every single server only executes one of the slice that is assigned to it. ElasticJob is aware of the number of servers in an almost-real-time manner, with the increment/decrement number of the servers, it re-assigns the job slices to the distributed servers, maximizing the efficiency as the increment of resources.

To execute the job in distributed servers, a job will be divided into multiple individual job items, one or some of which will be executed by the distributed servers.

For example, if a job is divided into 4 slices, and there are two servers to execute the job, then each server is assigned 2 slices, undertaking 50% of the workload, as follows.

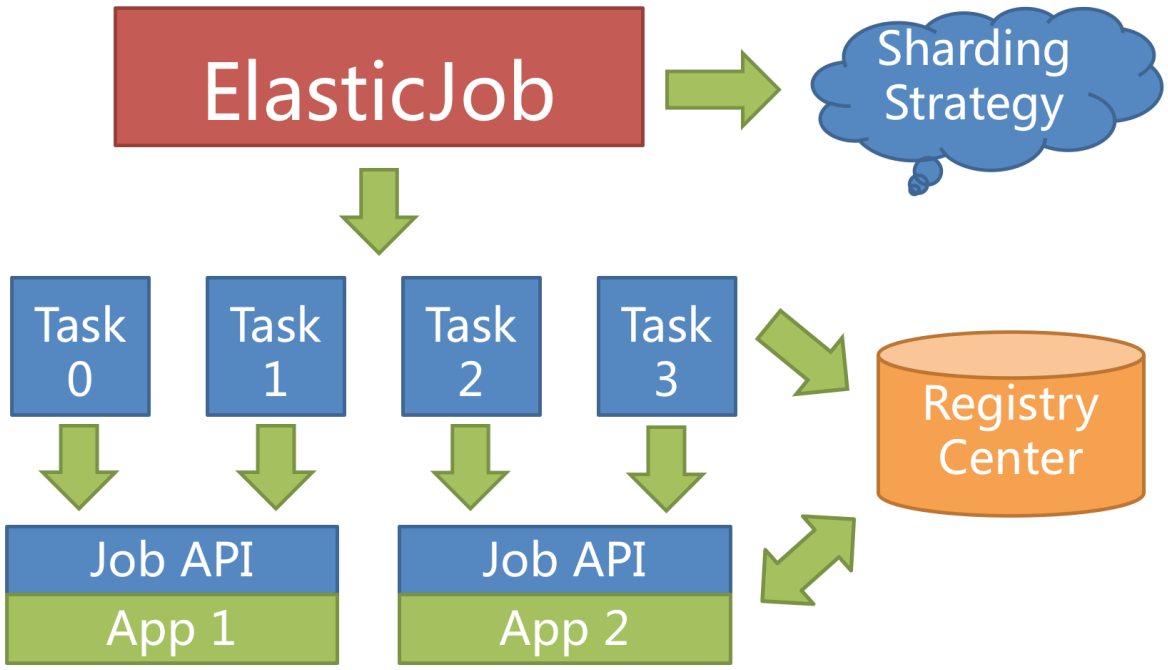

Figure1: Sharding Job

Sharding Item

ElasticJob doesn't directly provide the abilities to process the data, instead, it assigns the sharding items to the job servers, where the developers should process the sharding items and their business logic themselves. The sharding item is numeric type, in the range of [0, size(slices) - 1].

Customized sharding options

Customized sharding options can build a relationship with the sharding items, converting the sharding items' numbers to more readable business codes.

For example, to horizontally split the databases according to the regions, database A stores data from Beijing, database B stores data from Shanghai and database C stores data from Guangzhou. If we configure only by the sharding items' numbers, the developers need the knowledge that 0 represents Beijing, 1 represents Shanghai and 2 represents Guangzhou. Customized sharding options make the codes more readable, if we have customized options `0=Beijing,1=Shanghai,2=Guangzhou`, we can simply use `Beijing`, `Shanghai`, `Guangzhou` in the codes.

5.2.2 Maximize the usage of resources

ElasticJob provides a flexible way to maximize the throughput of the jobs. When new job server joins, ElasticJob will be aware of it from the registry, and will re-shard in the next scheduling process, the new server will undertake some of the job slices, as follows.

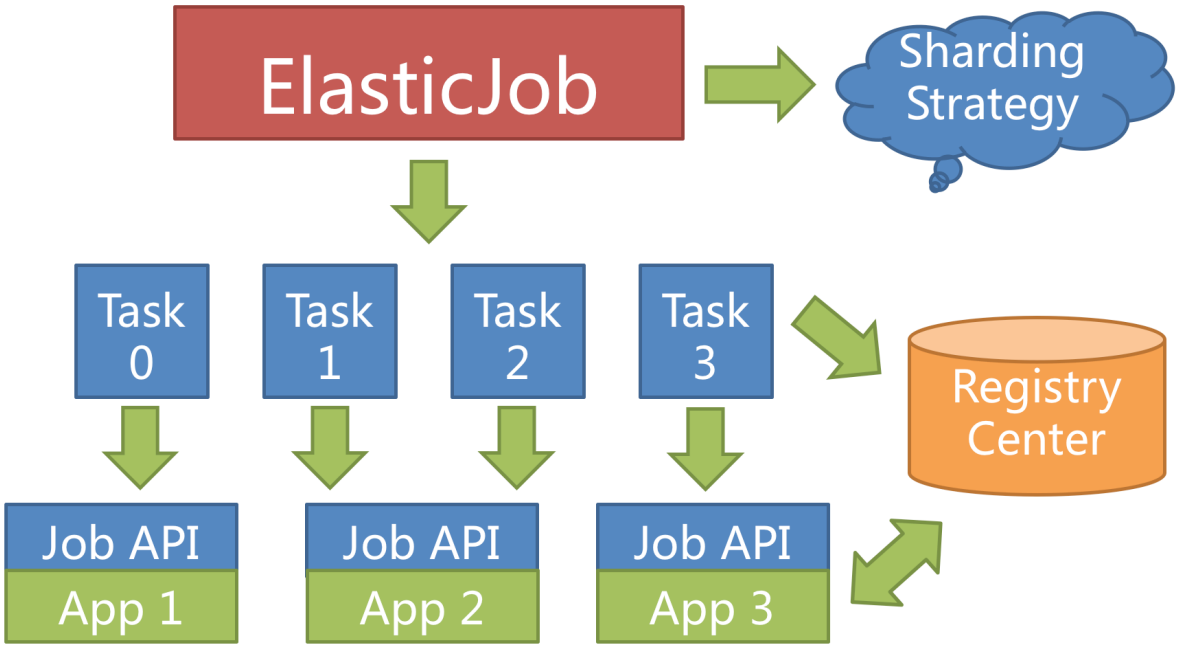

Figure2: scale out

Configuring a larger number of sharding items than the number of servers, or better, a multiplier of the number of servers, makes it more reasonably for the job to leverage the resources, and assign the sharding items dynamically.

For example, we have 10 sharding items and there are 3 servers, the number of sharding items are server A = 0,1,2,9; server B = 3,4,5; server C = 6,7,8. If the server C is down, then server A = 0,1,2,3,4 and B = 5,6,7,8,9, maximizing the throughput without losing any sharding item.

5.2.3 High Availability

When a server is down when executing a sharding item, the registry is also aware of that, and the sharding item will be transferred to another living server, thus achieve the goal of high availability. The unfinished job from a crashed server will be transferred and executed continuously, as follows.

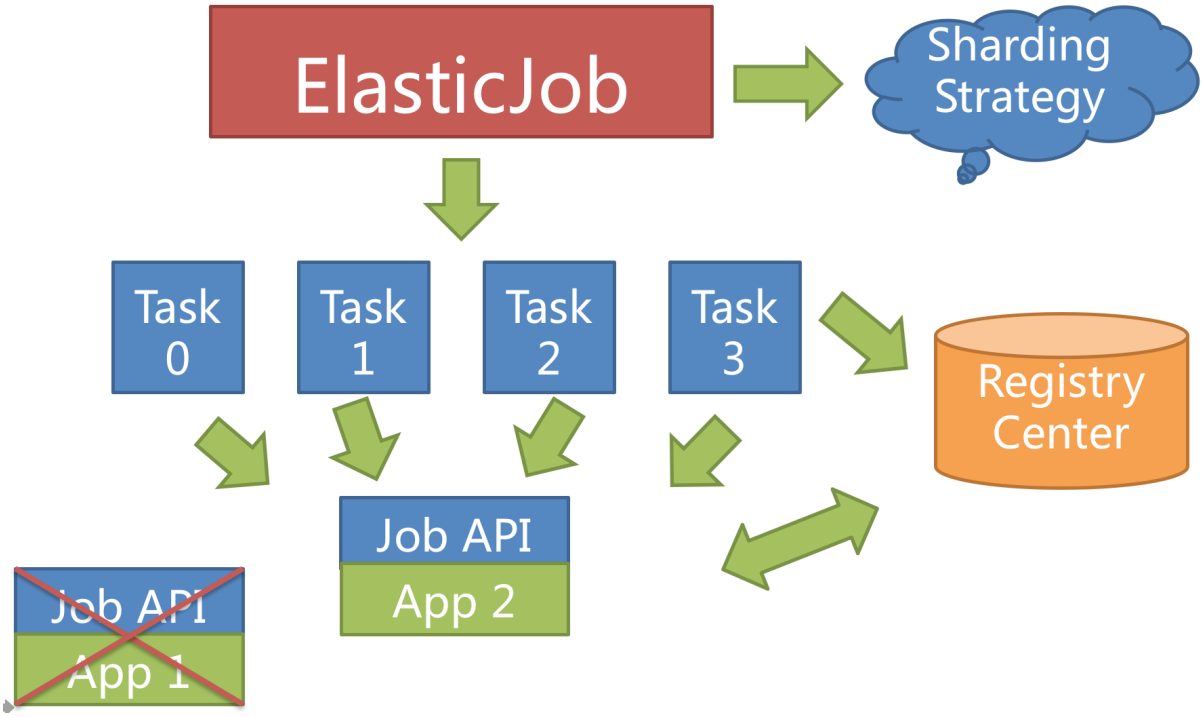

Figure3: HA

Setting the total number of sharding items to 1 and more than 1 servers to execute the jobs makes the job run in the mode of 1 master and n slaves. Once the servers that are executing jobs are down, the idle servers will take over the jobs and execute them in the next scheduling, or better, if the failover option is enabled, the idle servers can take over the failed jobs immediately.

5.2.4 ElasticJob-Lite Implementation Principle

ElasticJob-Lite does not have a job scheduling center node, but the programs based on the deployment job framework trigger the scheduling when the corresponding time point is reached. The registration center is only used for job registration and monitoring information storage. The main job node is only used to handle functions such as sharding and cleaning.

Elastic Distributed Implementation

- The first server went online to trigger the main server election. Once the main server goes offline, the election is triggered again, and the election process is blocked. Only when the main server election is completed, other tasks will be performed.
- When a job server goes online, it will automatically register the server information to the registry, and automatically update the server status when it goes offline.
- The re-sharding flag will be updated when the master node is elected, the server goes offline, and the total number of shards changes.
- When a scheduled task is triggered, if it needs to be sharded again, it will be sharded by the main server. The sharding process is blocked, and the task can be executed after the sharding ends. If the main server goes offline during the sharding process, the master server will be elected first and then perform sharding.
- From the previous description, in order to maintain the stability of the job runtime, only the sharding status will be marked during the running process, and the sharding will not be re-sharded. Sharding can only occur before the next task is triggered.
- Each execution of sharding will sort instances by server IP to ensure that the sharding result will not produce large fluctuations.
- Realize the failover function, actively grab the unallocated shards after a certain server is executed, and actively search for available servers to perform tasks after a certain server goes offline.

Registry Data Structure

The registration center creates a job name node under the defined namespace to distinguish different jobs, so once a job is created, the job name cannot be modified. If the name is modified, it will be regarded as a new job. There are 5 data sub-nodes under the job name node, namely config, instances, sharding, servers and leader.

config node

Job configuration information, stored in YAML format.

instances node

Job running instance information, the child node is the primary key of the current job running instance. The primary key of the job running instance is composed of the IP address and PID of the job running server. The primary keys of the job running instance are all ephemeral nodes, which are registered when the job instance is online and automatically cleaned up when the job instance is offline. The registry monitors the changes of these nodes to coordinate the sharding and high availability of distributed jobs. You can write TRIGGER in the job running instance node to indicate that the instance will be executed once immediately.

sharding node

Job sharding information. The child node is the sharding item sequence number, starting from zero and ending with the total number of shards minus one. The child node of the sharding item sequence number stores detailed information. The child node under each shard is used to control and record the running status of the shard. Node details description：

Child node name	Ephemeral node	Description
instance	NO	The primary key of the job running instance that executes the shard
running	YES	The running state of the shard item.Only valid when monitorExecution is configured
failover	YES	If the shard item is assigned to another job server by failover, this node value records the job server IP that executes the shard
misfire	NO	Whether to restart the missed task
disabled	NO	Whether to disable this shard

servers node

Job server information, the child node is the IP address of the job server. You can write DISABLED in the IP address node to indicate that the server is disabled. Under the new cloud-native architecture, the servers node is greatly weakened, only including controlling whether the server can be disabled. In order to achieve the core of the job more purely, the server function may be deleted in the future, and the ability to control whether the server is disabled should be delegated to the automated deployment system.

leader node

The master node information of the job server is divided into three sub-nodes: election, sharding and failover. They are used for master node election, sharding and failover processing respectively.

The leader node is an internally used node. If you are not interested in the principle of the job framework, you don’t need to pay attention to this node.

Childnodename	Ephemeralnode	Description
election:raw-latex:`instance`	YES	The IP address of the master node server.Once the node is deleted, a re-election will be triggered.All operations related to the master node will be blocked during the re-election process.
election:raw-latex:`latch`	NO	Distributed locks elected by the master nodeUsed for distributed locks of curator
sharding:raw-latex:`necessary`	NO	The flag for re-sharding. If the total number of shards changes, or the job server node goes online or offline or enabled/disabled, as well as the master node election, the re-sharded flag will be triggered. The master node is re-sharded without being interrupted in the middleThe sharding will not be triggered when the job is executed
sharding:raw-latex:`processing`	YES	The node held by the master node during sharding.If there is this node, all job execution will be blocked until the sharding ends.The ephemeral node will be deleted when the master node sharding is over or the master node crashes
failover:raw-latex:`items`:raw-latex:`sharding`item	NO	Once a job crashes, it will record to this node.When there is an idle job server, it will grab the job items that need to failover from this node
failover:raw-latex:`items`:raw-latex:`latch`	NO	Distributed locks used when allocating failover shard items. Used by curator distributed locks

5.3 Resource Assign

The resource allocation function is unique to ElasticJob-Cloud.

5.3.1 Execution mode

ElasticJob-Cloud is divided into two execution modes: transient and daemon execution.

Transient execution

The resources are released immediately after the execution of each job to ensure that the existing resources are used for staggered execution. Resource allocation and container startup both take up a certain amount of time, and resources may not be sufficient during job execution, so job execution may be delayed. Transient execution is suitable for jobs with long intervals, high resource consumption and no strict requirements on execution time.

Daemon execution

Whether it is running or waiting to run, it always occupies the allocated resources, which can save too many container startup and resource allocation costs, and is suitable for jobs with short intervals and stable resource requirements.

5.3.2 Scheduler

ElasticJob-Cloud is developed based on the Mesos Framework and is used for resource scheduling and application distribution. It needs to be started independently and provides services.

5.3.3 Job Application

Refers to the application after the job is packaged and deployed, and describes the basic information such as the CPU, memory, startup script, and application download path that are needed to start the job. Each job application can contain one or more jobs.

5.3.4 Job

That is, the specific tasks that are actually run share the same job ecology as ElasticJob-Lite. The job application must be registered before registering the job.

5.3.5 Resource

Refers to the CPU and memory required to start or run a job. Configuration in the job application dimension indicates the resources needed for the entire application to start; Configuration in the job dimension indicates the resources required for each job to run. The resources required for job startup are the sum of the resources required by the specified job application and the resources required by the job.

5.4 Failover

ElasticJob will not re-shard during this execution, but wait for the next scheduling before starting the re-sharding process. When the server is down during job execution, failover allows the unfinished task to be compensated and executed on another job node.

5.4.1 Concept

Failover is a temporary compensation execution mechanism for the currently executed job. When the next job is run, the current job allocation will be adjusted through resharding. For example, if the job is executed at an hourly interval, each execution will take 30 minutes. As shown below.

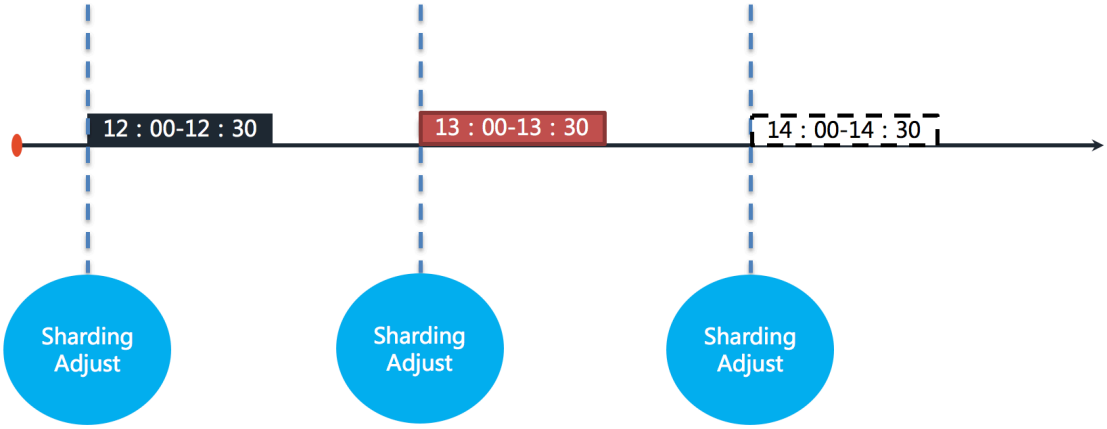

Figure4: Job

The figure shows that the jobs are executed at 12:00, 13:00 and 14:00 respectively. The current time point shown in the figure is the job execution at 13:00.

If one of the shard servers of the job goes down at 13:10, the remaining 20 minutes of the business that should be processed are not executed, and the next job can only be executed at 14:00. In other words, if failover is not turned on, there is a 50-minute idle period in this shard. As shown below.

After the failover is enabled, other ElasticJob servers can compensate for the execution of the sharding job after sensing the down job server. As shown below.

With sufficient resources, the job can still be executed completely at 13:30.

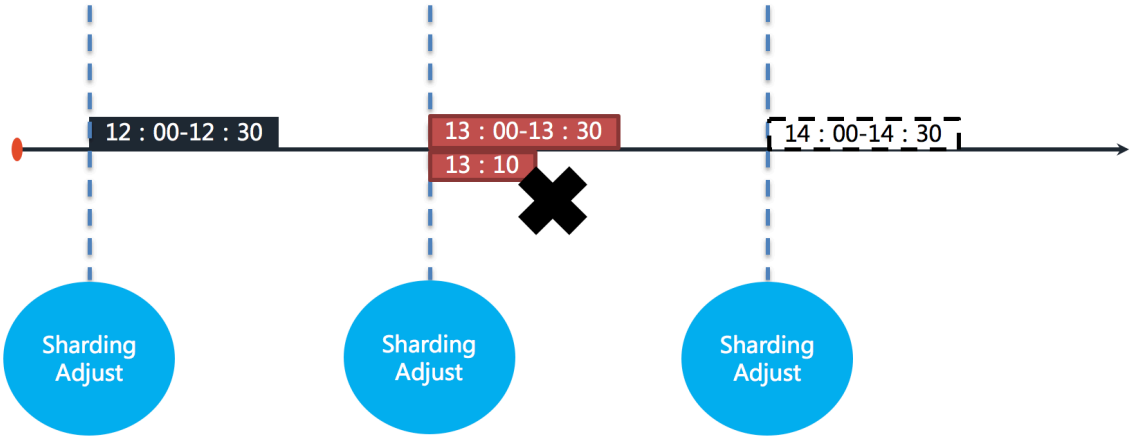

Figure5: Job Crash

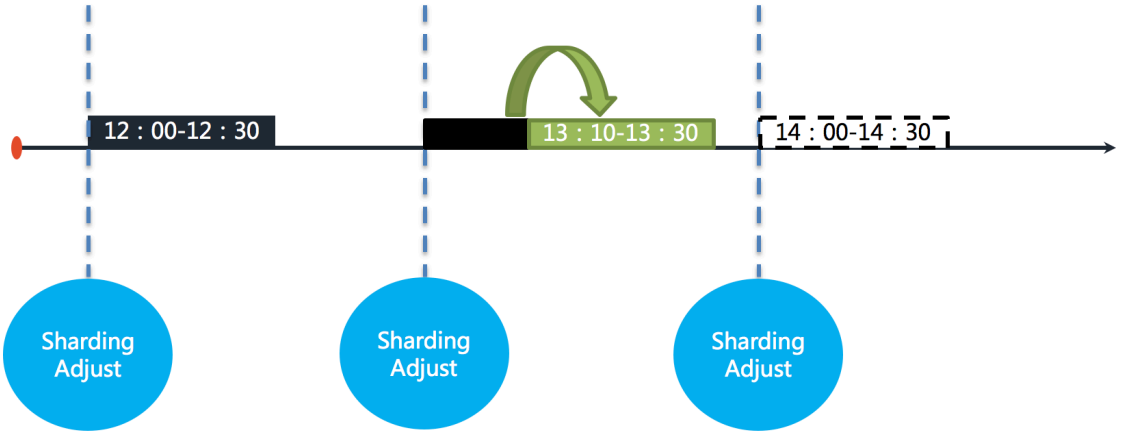

Figure6: Job Failover

5.4.2 Execution mechanism

When the job execution node goes down, the failover process will be triggered. ElasticJob determines the execution timing of the failover according to the different conditions of the distributed job execution when it is triggered。

Notification execution

When other servers perceive that a failover job needs to be processed, and the job server has completed this task, it will pull the items to be failed over in real time and start compensation execution. Also called real-time execution.

Enquiry execution

After the execution of this task, the job service will inquire about the failover items to be executed from the registry, and if there are any, the compensation execution will start. Also called asynchronous execution.

5.4.3 Scenarios:

With the failover enabled, ElasticJob will monitor the execution status of each shard of the job and write it to the registry for other nodes to perceive.

In a job scenario that takes a long time to run and has a long interval, failover is an effective means to improve the real-time operation of the job; For short-interval jobs, a large number of network communications with the registry will be generated, which will affect the performance of the cluster; Moreover, short-interval jobs do not necessarily pay attention to the real-time performance of a single job. You can use the re-shard of the next job execution to make all the items execute correctly. Therefore, it is not recommended to enable failover for short-interval jobs.

Another thing to note is that the idempotence of the job itself is a prerequisite to ensure the correctness of failover.

5.5 Misfire

ElasticJob does not allow jobs to be executed at the same time. When the execution time of a job exceeds its running interval, re-executing the missed task can ensure that the job continues to execute the overdue job after completing the last task.

5.5.1 Concept

The misfire function enables the overdue tasks to be executed immediately after the completion of the previous tasks. For example, if the job is executed at an hourly interval, each execution will take 30 minutes. As shown below.

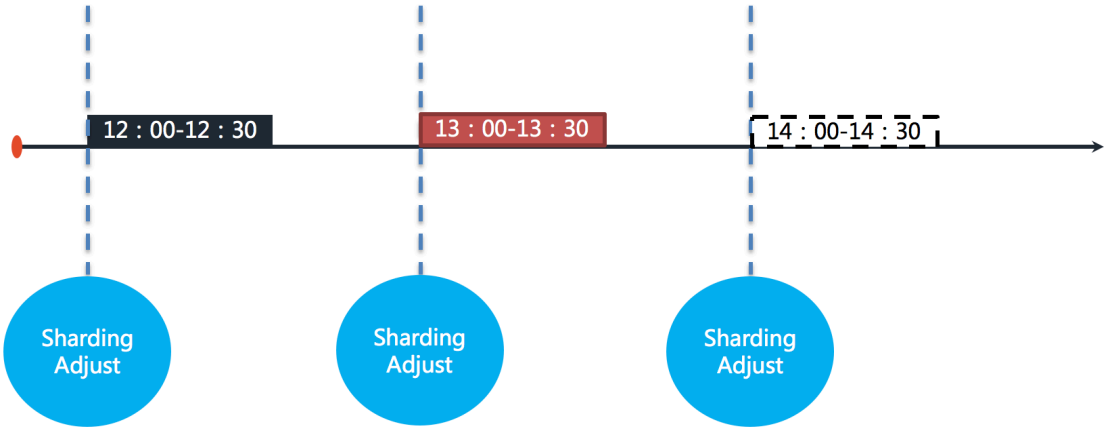

Figure7: Job

The figure shows that the jobs are executed at 12:00, 13:00 and 14:00 respectively. The current time point shown in the figure is the job execution at 13:00.

If the job executed at 12:00 is finished at 13:10, then the job that should have been triggered by 13:00 missed the trigger time and needs to wait until the next job trigger at 14:00. As shown below.

Figure8: Job Missed

After the misfire is enabled, ElasticJob will trigger the execution of the missed job immediately after the last job is executed. As shown below.

Missed jobs between 13:00 and 14:00 will be executed again.

Figure9: Job Misfire

5.5.2 Scenarios

In a job scenario that takes a long time to run and has a long interval, misfire is an effective means to improve the real-time operation of the job; For short-interval jobs that do not necessarily pay attention to the real-time performance of a single job, it is not necessary to turn on the misfire to re-execute.

5.6 Job Open Ecosystem

Flexible customized jobs is the most important design change in ElasticJob 3.x . The new version is based on the design concept of the Apache ShardingSphere pluggable architecture, and the new Job API was created. It is intended to enable developers to expand the types of jobs in a more convenient and isolated way, and create an ecosystem of ElasticJob jobs.

While ElasticJob provides functions such as elastic scaling and distributed management of jobs, it does not limit the types of jobs. It uses flexible job APIs to decouple jobs into job interfaces and actuator interfaces. Users can customize new job types, such as script execution, HTTP service execution, big data jobs, file jobs, etc. At present, ElasticJob has built-in simple jobs, data flow jobs, and script execution jobs, and has completely opened up the extension interface. Developers can introduce new job types through SPI, and they can easily give back to the community.

5.6.1 Job interface

ElasticJob jobs can be divided into two types: `Class-based Jobs` and `Type-based Jobs`.

`Class-based Jobs` are directly used by developers, who need to implement the job interface to realize business logic. Typical representatives: Simple type, Dataflow type. `Type-based Jobs` only need to provide the type name, developers do not need to implement the job interface, but use it through external configuration. Typical representatives: Script type, HTTP type (Since 3.0.0-beta).

5.6.2 Actuator interface

It is used to execute user-defined job interfaces and weave into the ElasticJob ecosystem through Java's SPI mechanism.

6 User Manual

This chapter describes how to use projects of ElasticJob: ElasticJob-Lite and ElasticJob-Cloud.

6.1 ElasticJob-Lite

6.1.1 Introduction

ElasticJob-Lite is a lightweight, decentralized solution that provides distributed task sharding services.

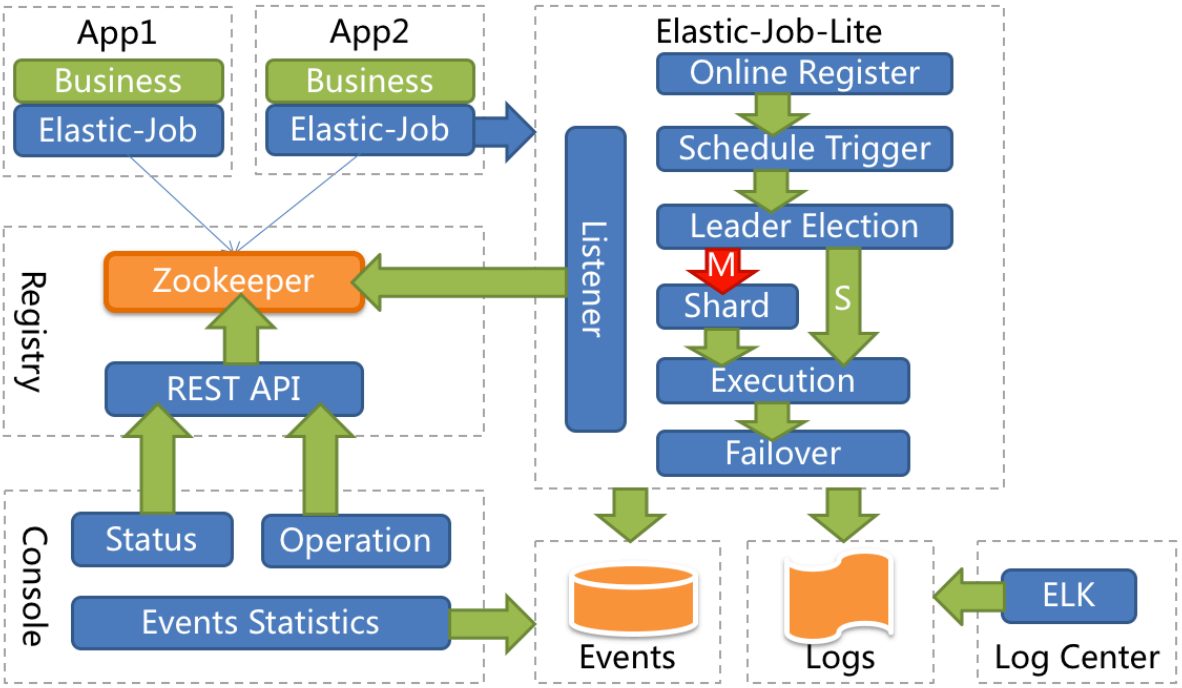

Figure1: ElasticJob-Lite Architecture

6.1.2 Comparison

	ElasticJob-Lite	*ElasticJob-Cloud*
Decentralization	`Yes`	No
Resource Assign	`No`	Yes
Job Execution	`Daemon`	Daemon + Transient
Deploy Dependency	`ZooKeeper`	ZooKeeper + Mesos

The advantages of ElasticJob-Lite are no centralized design and less external dependence, which is suitable for business application with stable resource allocation.

6.1.3 Usage

This chapter will introduce the use of ElasticJob-Lite. Please refer to Example for more details.

Job API

ElasticJob-Lite can use for native Java, Spring Boot Starter and Spring namespace. This chapter will introduce how to use them.

Job Development

ElasticJob-Lite and ElasticJob-Cloud provide a unified job interface, developers need to develop business jobs only once, and then they can be deployed to different environments according to different configurations and deployments.

ElasticJob has two kinds of job types: Class-based job and Type-based job. Class-based jobs require developers to weave business logic by implementing interfaces; Type-based jobs don't need coding, just need to provide the corresponding configuration.

The method parameter `shardingContext` of the class-based job interface contains job configuration, slice and runtime information. Through methods such as `getShardingTotalCount()`, `getShardingItem()`, user can obtain the total number of shards, the serial number of the shards running on the job server, etc.

ElasticJob provides two class-based job types which are `Simple` and `Dataflow`; and also provides a type-based job which is `Script`. Users can extend job types by implementing the SPI interface.

Simple Job

It means simple implementation, without any encapsulation type. Need to implement `SimpleJob` interface. This interface only provides a single method for coverage, and this method will be executed periodically. It is similar to Quartz's native interface, but provides functions such as elastic scaling and slice.

```
public class MyElasticJob implements SimpleJob {

    @Override
    public void execute(ShardingContext context) {
        switch (context.getShardingItem()) {
            case 0:
                // do something by sharding item 0
                break;
            case 1:
                // do something by sharding item 1
                break;
            case 2:
                // do something by sharding item 2
                break;
            // case n: ...
        }
    }
}
```

Dataflow Job

For processing data flow, need to implement `DataflowJob` interface. This interface provides two methods for coverage, which are used to fetch (fetchData) and process (processData) data.

```
public class MyElasticJob implements DataflowJob<Foo> {

    @Override
    public List<Foo> fetchData(ShardingContext context) {
        switch (context.getShardingItem()) {
            case 0:
                List<Foo> data = // get data from database by sharding item 0
                return data;
            case 1:
                List<Foo> data = // get data from database by sharding item 1
                return data;
            case 2:
                List<Foo> data = // get data from database by sharding item 2
                return data;
            // case n: ...
        }
```

```
    }

    @Override
    public void processData(ShardingContext shardingContext, List<Foo> data) {
        // process data
        // ...
    }
}
```

Streaming

Streaming can be enabled or disabled through the property `streaming.process`.

If streaming is enabled, the job will stop fetching data only when the return value of the `fetchData` method is null or the collection is empty, otherwise the job will continue to run; If streaming is disabled, the job will execute the `fetchData` and `processData` methods only once during each job execution, and then the job will be completed immediately.

If use the streaming job to process data, it is recommended to update its status after the `processData` method being executed, to avoid being fetched again by the method `fetchData`, so that the job never stops.

Script job

Support all types of scripts such as `shell`, `python`, `perl`. The script to be executed can be configured through the property `script.command.line`, without coding. The script path can contain parameters, after the parameters are passed, the job framework will automatically append the last parameter as the job runtime information.

The script example is as follows:

```
#!/bin/bash
echo sharding execution context is $*
```

When the job runs, it will output:

```
sharding execution context is {"jobName":"scriptElasticDemoJob","shardingTotalCount
":10,"jobParameter":"","shardingItem":0,"shardingParameter":"A"}
```

HTTP job (Since 3.0.0-beta)

The http information to be requested can be configured through the properties of `http.url`, `http.method`, `http.data`, etc. Sharding information is transmitted in the form of Header, the key is `shardingContext`, and the value is in json format.

```
public class HttpJobMain {

    public static void main(String[] args) {

        new ScheduleJobBootstrap(regCenter, "HTTP", JobConfiguration.newBuilder(
"javaHttpJob", 1)
                .setProperty(HttpJobProperties.URI_KEY, "http://xxx.com/execute")
                .setProperty(HttpJobProperties.METHOD_KEY, "POST")
                .setProperty(HttpJobProperties.DATA_KEY, "source=ejob")
                .cron("0/5 * * * * ?").shardingItemParameters("0=Beijing").
build()).schedule();
    }
}
```

```
@Controller
@Slf4j
public class HttpJobController {

    @RequestMapping(path = "/execute", method = RequestMethod.POST)
    public void execute(String source, @RequestHeader String shardingContext) {
        log.info("execute from source : {}, shardingContext : {}", source,
shardingContext);
    }
}
```

When the job runs, it will output:

```
execute from source : ejob, shardingContext : {"jobName":"scriptElasticDemoJob",
"shardingTotalCount":3,"jobParameter":"","shardingItem":0,"shardingParameter":
"Beijing"}
```

Use Java API

Job configuration

ElasticJob-Lite uses the builder mode to create job configuration objects. The code example is as follows:

```
JobConfiguration jobConfig = JobConfiguration.newBuilder("myJob", 3).cron("0/5 * *
* * ?").shardingItemParameters("0=Beijing,1=Shanghai,2=Guangzhou").build();
```

Job start

ElasticJob-Lite scheduler is divided into two types: timed scheduling and one-time scheduling. Each scheduler needs three parameters: registry configuration, job object (or job type), and job configuration when it starts.

Timed scheduling

```
public class JobDemo {

    public static void main(String[] args) {
        // Class-based Scheduling Jobs
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createJobConfiguration()).schedule();
        // Type-based Scheduling Jobs
        new ScheduleJobBootstrap(createRegistryCenter(), "MY_TYPE",
createJobConfiguration()).schedule();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(new
ZookeeperConfiguration("zk_host:2181", "elastic-job-demo"));
        regCenter.init();
        return regCenter;
    }

    private static JobConfiguration createJobConfiguration() {
        // Create job configuration
        ...
    }
}
```

One-Off scheduling

```
public class JobDemo {

    public static void main(String[] args) {
        OneOffJobBootstrap jobBootstrap = new
OneOffJobBootstrap(createRegistryCenter(), new MyJob(), createJobConfiguration());
        // One-time scheduling can be called multiple times
        jobBootstrap.execute();
        jobBootstrap.execute();
        jobBootstrap.execute();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
```

```
        CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(new
ZookeeperConfiguration("zk_host:2181", "elastic-job-demo"));
        regCenter.init();
        return regCenter;
    }

    private static JobConfiguration createJobConfiguration() {
        // Create job configuration
        ...
    }
}
```

Job Dump

Using ElasticJob may meet some distributed problem which is not easy to observe.

Because of developer can not debug in production environment, ElasticJob provide dump command to export job runtime information for debugging.

Please refer to Operation Manual for more details.

The example below is how to configure spring namespace for open listener port to dump.

```
public class JobMain {

    public static void main(final String[] args) {
        SnapshotService snapshotService = new SnapshotService(regCenter, 9888).
listen();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center
    }
}
```

Configuration error handler strategy

In the process of using ElasticJob-Lite, when the job is abnormal, the following error handling strategies can be used.

Error handler strategy name	*Description*	• Built-in*	• Default*	*Extra config*
Log Strategy	Log error and do not interrupt job	Yes	Yes	
Throw Strategy	Throw system exception and interrupt job	Yes		
Ignore Strategy	Ignore exception and do not interrupt job	Yes		
Email Notification Strategy	Send email message notification and do not interrupt job			Yes
Wechat Enterprise Notification Strategy	Send wechat message notification and do not interrupt job			Yes
Dingtalk Notification Strategy	Send dingtalk message notification and do not interrupt job			Yes

Log Strategy

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of log strategy
        return JobConfiguration.newBuilder("myScheduleJob", 3).cron("0/5 * * * * ?
").jobErrorHandlerType("LOG").build();
```

```
    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of log strategy
        return JobConfiguration.newBuilder("myOneOffJob", 3).jobErrorHandlerType(
"LOG").build();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center
        ...
    }
}
```

Throw Strategy

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of throw strategy.
        return JobConfiguration.newBuilder("myScheduleJob", 3).cron("0/5 * * * * ?
").jobErrorHandlerType("THROW").build();
    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of throw strategy
        return JobConfiguration.newBuilder("myOneOffJob", 3).jobErrorHandlerType(
"THROW").build();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center
        ...
    }
}
```

Ignore Strategy

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of ignore strategy.
        return JobConfiguration.newBuilder("myScheduleJob", 3).cron("0/5 * * * * ?
").jobErrorHandlerType("IGNORE").build();
    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of ignore strategy.
        return JobConfiguration.newBuilder("myOneOffJob", 3).jobErrorHandlerType(
"IGNORE").build();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center.
        ...
    }
}
```

Email Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-email</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs
```

```
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of email notification
strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myScheduleJob",
3).cron("0/5 * * * * ?").jobErrorHandlerType("EMAIL").build();
        setEmailProperties(jobConfig);
        return jobConfig;

    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of email notification
strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myOneOffJob", 3).
jobErrorHandlerType("EMAIL").build();
        setEmailProperties(jobConfig);
        return jobConfig;
    }

    private static void setEmailProperties(final JobConfiguration jobConfig) {
        // Set the mail configuration.
        jobConfig.getProps().setProperty(EmailPropertiesConstants.HOST, "host");
        jobConfig.getProps().setProperty(EmailPropertiesConstants.PORT, "465");
        jobConfig.getProps().setProperty(EmailPropertiesConstants.USERNAME,
"username");
        jobConfig.getProps().setProperty(EmailPropertiesConstants.PASSWORD,
"password");
        jobConfig.getProps().setProperty(EmailPropertiesConstants.FROM, "from@xxx.
xx");
        jobConfig.getProps().setProperty(EmailPropertiesConstants.TO, "to1@xxx.xx,
to1@xxx.xx");
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center.
        ...
    }
}
```

Wechat Enterprise Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-wechat</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs.
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs.
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of wechat enterprise
notification strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myScheduleJob",
3).cron("0/5 * * * * ?").jobErrorHandlerType("WECHAT").build();
        setWechatProperties(jobConfig);
        return jobConfig;

    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of wechat enterprise
notification strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myOneOffJob", 3).
jobErrorHandlerType("WECHAT").build();
        setWechatProperties(jobConfig);
        return jobConfig;
    }

    private static void setWechatProperties(final JobConfiguration jobConfig) {
        // Set the configuration for the enterprise wechat.
        jobConfig.getProps().setProperty(WechatPropertiesConstants.WEBHOOK, "you_
webhook");
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
```

```
        // create registry center.
        ...
    }
}
```

Dingtalk Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-dingtalk</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
public class JobDemo {

    public static void main(String[] args) {
        //  Scheduling Jobs.
        new ScheduleJobBootstrap(createRegistryCenter(), new MyJob(),
createScheduleJobConfiguration()).schedule();
        // One-time Scheduling Jobs.
        new OneOffJobBootstrap(createRegistryCenter(), new MyJob(),
createOneOffJobConfiguration()).execute();
    }

    private static JobConfiguration createScheduleJobConfiguration() {
        // Create scheduling job configuration, and the use of dingtalk
notification strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myScheduleJob",
3).cron("0/5 * * * * ?").jobErrorHandlerType("DINGTALK").build();
        setDingtalkProperties(jobConfig);
        return jobConfig;

    }

    private static JobConfiguration createOneOffJobConfiguration() {
        // Create one-time job configuration, and the use of dingtalk notification
strategy.
        JobConfiguration jobConfig = JobConfiguration.newBuilder("myOneOffJob", 3).
jobErrorHandlerType("DINGTALK").build();
        setDingtalkProperties(jobConfig);
        return jobConfig;
    }
```

```
    private static void setDingtalkProperties(final JobConfiguration jobConfig) {
        // Set the configuration of the dingtalk.
        jobConfig.getProps().setProperty(DingtalkPropertiesConstants.WEBHOOK, "you_
webhook");
        jobConfig.getProps().setProperty(DingtalkPropertiesConstants.KEYWORD, "you_
keyword");
        jobConfig.getProps().setProperty(DingtalkPropertiesConstants.SECRET, "you_
secret");
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        // create registry center.
        ...
    }
}
```

Use Spring Boot Starter

ElasticJob-Lite provides a customized Spring Boot Starter, which can be used in conjunction with Spring Boot. Developers are free from configuring CoordinatorRegistryCenter, JobBootstrap by using ElasticJob Spring Boot Starter. What developers need to solve distributed scheduling problem are job implementations with a little configuration.

Job configuration

Implements ElasticJob

Job implementation is similar to other usage of ElasticJob. The difference is that jobs will be registered into the Spring IoC container.

Thread-Safety Issue

Bean is singleton by default. Consider setting Bean Scope to `prototype` if the instance of ElasticJob would be used by more than a JobBootstrap.

```
@Component
public class SpringBootDataflowJob implements DataflowJob<Foo> {

    @Override
    public List<Foo> fetchData(final ShardingContext shardingContext) {
        // fetch data
    }

    @Override
    public void processData(final ShardingContext shardingContext, final List<Foo>
data) {
        // process data
```

```
    }
}
```

Configure CoordinateRegistryCenter and Jobs

Configure the Zookeeper which will be used by ElasticJob via configuration files.

`elasticjob.jobs` is a Map. Using key as job name. Specific job type and configuration in value. The Starter will create instances of `OneOffJobBootstrap` or `ScheduleJobBootstrap` and register them into the Spring IoC container automatically.

Configuration reference:

```
elasticjob:
  regCenter:
    serverLists: localhost:6181
    namespace: elasticjob-lite-springboot
  jobs:
    dataflowJob:
      elasticJobClass: org.apache.shardingsphere.elasticjob.dataflow.job.
DataflowJob
      cron: 0/5 * * * * ?
      shardingTotalCount: 3
      shardingItemParameters: 0=Beijing,1=Shanghai,2=Guangzhou
    scriptJob:
      elasticJobType: SCRIPT
      cron: 0/10 * * * * ?
      shardingTotalCount: 3
      props:
        script.command.line: "echo SCRIPT Job: "
```

Job Start

Schedule Job

Just start Spring Boot Starter directly. The schedule jobs will startup when the Spring Boot Application is started.

One-off Job

When to execute OneOffJob is up to you. Developers can inject the `OneOffJobBootstrap` bean into where they plan to invoke. Trigger the job by invoking `execute()` method manually.

The bean name of `OneOffJobBootstrap` is specified by property "jobBootstrapBeanName", Please refer to Spring Boot Starter Configuration.

```
elasticjob:
  jobs:
    myOneOffJob:
      jobBootstrapBeanName: myOneOffJobBean
      ....
```

```
@RestController
public class OneOffJobController {

    // Inject via "@Resource"
    @Resource(name = "myOneOffJobBean")
    private OneOffJobBootstrap myOneOffJob;

    @GetMapping("/execute")
    public String executeOneOffJob() {
        myOneOffJob.execute();
        return "{\"msg\":\"OK\"}";
    }

    // Inject via "@Autowired"
    @Autowired
    @Qualifier(name = "myOneOffJobBean")
    private OneOffJobBootstrap myOneOffJob2;

    @GetMapping("/execute2")
    public String executeOneOffJob2() {
        myOneOffJob2.execute();
        return "{\"msg\":\"OK\"}";
    }
}
```

Configuration error handler strategy

In the process of using ElasticJob-Lite, when the job is abnormal, the following error handling strategies can be used.

Error handler strategy name	*Description*	• Built-in*	• Default*	*Extra config*
Log Strategy	Log error and do not interrupt job	Yes	Yes	
Throw Strategy	Throw system exception and interrupt job	Yes		
Ignore Strategy	Ignore exception and do not interrupt job	Yes		
Email Notification Strategy	Send email message notification and do not interrupt job			Yes
Wechat Enterprise Notification Strategy	Send wechat message notification and do not interrupt job			Yes
Dingtalk Notification Strategy	Send dingtalk message notification and do not interrupt job			Yes

Log Strategy

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: LOG
```

Throw Strategy

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: THROW
```

Ignore Strategy

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: IGNORE
```

Email Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-email</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: EMAIL
  props:
    email:
      host: host
      port: 465
      username: username
      password: password
      useSsl: true
      subject: ElasticJob error message
      from: from@xxx.xx
      to: to1@xxx.xx,to2@xxx.xx
```

```
      cc: cc@xxx.xx
      bcc: bcc@xxx.xx
      debug: false
```

Wechat Enterprise Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-wechat</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: WECHAT
  props:
    wechat:
      webhook: you_webhook
      connectTimeout: 3000
      readTimeout: 5000
```

Dingtalk Notification Strategy

Please refer to here for more details.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-dingtalk</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

```
elasticjob:
  regCenter:
    ...
  jobs:
    ...
    jobErrorHandlerType: DINGTALK
```

```
  props:
    dingtalk:
       webhook: you_webhook
       keyword: you_keyword
       secret: you_secret
       connectTimeout: 3000
       readTimeout: 5000
```

Use Spring Namespace

ElasticJob-Lite provides a custom Spring namespace, which can be used with the Spring. Through the way of DI (Dependency Injection), developers can easily use data sources and other objects that managed by the Spring container in their jobs, and use placeholders to get values from property files.

Job Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:elasticjob="http://shardingsphere.apache.org/schema/elasticjob"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://shardingsphere.apache.org/schema/elasticjob
                        http://shardingsphere.apache.org/schema/elasticjob/
elasticjob.xsd
                        ">
    <!-- Configure registry center for job -->
    <elasticjob:zookeeper id="regCenter" server-lists="yourhost:2181" namespace=
"my-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000"
max-retries="3" />

    <!-- Configure job java bean -->
    <bean id="myJob" class="xxx.MyJob">
        <property name="fooService" ref="xxx.FooService" />
    </bean>

    <!-- Configure job scheduler base on java bean -->
    <elasticjob:job id="${myJob.id}" job-ref="myJob" registry-center-ref="regCenter
" sharding-total-count="${myJob.shardingTotalCount}" cron="${myJob.cron}" />

   <!-- Configure job scheduler base on type -->
    <elasticjob:job id="${myScriptJob.id}" job-type="SCRIPT" registry-center-ref=
"regCenter" sharding-total-count="${myScriptJob.shardingTotalCount}" cron="$
{myScriptJob.cron}">
        <props>
```

```
            <prop key="script.command.line">${myScriptJob.scriptCommandLine}</prop>
        </props>
    </elasticjob:job>
</beans>
```

Job Start

Schedule Job

If the Spring container start, the `XML` that configures the Spring namespace will be loaded, and the job will be automatically started.

One-off Job

When to execute OneOffJob is up to you. Developers can inject the `OneOffJobBootstrap` bean into where they plan to invoke. Trigger the job by invoking `execute()` method manually.

```
<bean id="oneOffJob" class="org.apache.shardingsphere.elasticjob.lite.example.job.
simple.SpringSimpleJob" />
<elasticjob:job id="oneOffJobBean" job-ref="oneOffJob" ...  />
```

```
public final class SpringMain {
    public static void main(final String[] args) {
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("classpath:META-INF/application-context.xml");
        OneOffJobBootstrap oneOffJobBootstrap = context.getBean("oneOffJobBean",
OneOffJobBootstrap.class);
        oneOffJobBootstrap.execute();
    }
}
```

Job Dump

Using ElasticJob may meet some distributed problem which is not easy to observe.

Because of developer can not debug in production environment, ElasticJob provide dump command to export job runtime information for debugging.

Please refer to Operation Manual for more details.

The example below is how to configure SnapshotService for open listener port to dump.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:elasticjob="http://shardingsphere.apache.org/schema/elasticjob"
```

```
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/elasticjob
                           http://shardingsphere.apache.org/schema/elasticjob/
elasticjob.xsd
                         ">
    <!--Create registry center -->
    <elasticjob:zookeeper id="regCenter" server-lists="yourhost:2181" namespace=
"dd-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000"
max-retries="3" />

    <!--Configure the task snapshot export service -->
    <elasticjob:snapshot id="jobSnapshot" registry-center-ref="regCenter" dump-
port="9999" />
</beans>
```

Configuration error handler strategy

In the process of using ElasticJob-Lite, when the job is abnormal, the following error handling strategies can be used.

Error handler strategy name	*Description*	• Built-in*	• Default*	*E xtra con fig*
Log Strategy	Log error and do not interrupt job	Y e s	Y e s	
Throw Strategy	Throw system exception and interrupt job	Y e s		
Ignore Strategy	Ignore exception and do not interrupt job	Y e s		
Email Notification Strategy	Send email message notification and do not interrupt job			Yes
Wechat Enterprise Notification Strategy	Send wechat message notification and do not interrupt job			Yes
Dingtalk Notification Strategy	Send dingtalk message notification and do not interrupt job			Yes

The following example shows how to configure the error-handling policy through the Spring namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:elasticjob="http://shardingsphere.apache.org/schema/elasticjob"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/elasticjob
                           http://shardingsphere.apache.org/schema/elasticjob/
elasticjob.xsd
                         ">
    <!-- Log Strategy -->
    <elasticjob:job  ... job-error-handler-type="LOG" />

    <!-- Throw Strategy -->
    <elasticjob:job  ... job-error-handler-type="THROW" />

    <!-- Ignore Strategy -->
    <elasticjob:job  ... job-error-handler-type="IGNORE" />

    <!-- Email Notification Strategy  -->
    <elasticjob:job ... job-error-handler-type="EMAIL">
        <props>
            <prop key="email.host">${host}</prop>
            <prop key="email.port">${port}</prop>
            <prop key="email.username">${username}</prop>
            <prop key="email.password">${password}</prop>
            <prop key="email.useSsl">${useSsl}</prop>
            <prop key="email.subject">${subject}</prop>
            <prop key="email.from">${from}</prop>
            <prop key="email.to">${to}</prop>
            <prop key="email.cc">${cc}</prop>
            <prop key="email.bcc">${bcc}</prop>
            <prop key="email.debug">${debug}</prop>
        </props>
    </elasticjob:job>

    <!-- Wechat Enterprise Notification Strategy -->
    <elasticjob:job  ... job-error-handler-type="WECHAT">
        <props>
            <prop key="wechat.webhook">${webhook}</prop>
            <prop key="wechat.connectTimeoutMilliseconds">$
{connectTimeoutMilliseconds}</prop>
            <prop key="wechat.readTimeoutMilliseconds">${readTimeoutMilliseconds}</
prop>
        </props>
```

```
    </elasticjob:job>

    <!-- Dingtalk Notification Strategy  -->
    <elasticjob:job  ... job-error-handler-type="DINGTALK">
        <props>
            <prop key="dingtalk.webhook">${webhook}</prop>
            <prop key="dingtalk.keyword">${keyword}</prop>
            <prop key="dingtalk.secret">${secret}</prop>
            <prop key="dingtalk.connectTimeoutMilliseconds">$
{connectTimeoutMilliseconds}</prop>
            <prop key="dingtalk.readTimeoutMilliseconds">${readTimeoutMilliseconds}
</prop>
        </props>
    </elasticjob:job>
</beans>
```

Job Listener

ElasticJob-Lite provides job listeners, which are used to perform monitoring methods before and after task execution. Listeners are divided into regular listeners executed by each job node and distributed listeners executed by only a single node in a distributed scenario. This chapter will introduce how to use them in detail.

After the job dependency (DAG) function is developed, the job listener function may be considered to be deleted.

Listener Development

Common Listener

If the job processes the files of the job server and deletes the files after the processing is completed, consider using each node to perform the cleaning task. This type of task is simple to implement, and there is no need to consider whether the global distributed task is completed. You should try to use this type of listener.

```
public class MyJobListener implements ElasticJobListener {

    @Override
    public void beforeJobExecuted(ShardingContexts shardingContexts) {
        // do something ...
    }

    @Override
    public void afterJobExecuted(ShardingContexts shardingContexts) {
        // do something ...
    }
```

```
    @Override
    public String getType() {
        return "simpleJobListener";
    }
}
```

Distributed Listener

If the job processes database data, only one node needs to complete the data cleaning task after the processing is completed. This type of task is complicated to process and needs to synchronize the status of the job in a distributed environment. Timeout settings are provided to avoid deadlocks caused by job out of sync. It should be used with caution.

```
public class MyDistributeOnceJobListener extends
AbstractDistributeOnceElasticJobListener {

    public TestDistributeOnceElasticJobListener(long startTimeoutMills, long
completeTimeoutMills) {
        super(startTimeoutMills, completeTimeoutMills);
    }

    @Override
    public void doBeforeJobExecutedAtLastStarted(ShardingContexts shardingContexts)
{
        // do something ...
    }

    @Override
    public void doAfterJobExecutedAtLastCompleted(ShardingContexts
shardingContexts) {
        // do something ...
    }

    @Override
    public String getType() {
        return "distributeOnceJobListener";
    }
}
```

Add SPI implementation

Put JobListener implementation to module infra-common, resources/META-INF/services/org.apache.shardingsphere.elasticjob.infra.listener.ElasticJobListener

Use Java API

Common Listener

```
public class JobMain {

    public static void main(String[] args) {
        new ScheduleJobBootstrap(createRegistryCenter(), createJobConfiguration()).
schedule();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(new
ZookeeperConfiguration("zk_host:2181", "elastic-job-demo"));
        regCenter.init();
        return regCenter;
    }

    private static JobConfiguration createJobConfiguration() {
        JobConfiguration jobConfiguration = JobConfiguration.newBuilder("test", 2)
                                        .jobListenerTypes("simpleListener",
"distributeListener").build();
    }
}
```

Distributed Listener

```
public class JobMain {

    public static void main(String[] args) {
        new ScheduleJobBootstrap(createRegistryCenter(), createJobConfiguration()).
schedule();
    }

    private static CoordinatorRegistryCenter createRegistryCenter() {
        CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(new
ZookeeperConfiguration("zk_host:2181", "elastic-job-demo"));
        regCenter.init();
        return regCenter;
    }
```

```
    private static JobConfiguration createJobConfiguration() {
        JobConfiguration jobConfiguration = JobConfiguration.newBuilder("test", 2)
                                                .jobListenerTypes("simpleListener",
"distributeListener").build();
    }
}
```

Use Spring Namespace

Listener configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:elasticjob="http://shardingsphere.apache.org/schema/elasticjob"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://shardingsphere.apache.org/schema/elasticjob
                        http://shardingsphere.apache.org/schema/elasticjob/
elasticjob.xsd
                        ">
    <!-- Configuration job registration center -->
    <elasticjob:zookeeper id="regCenter" server-lists="yourhost:2181" namespace=
"my-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000"
max-retries="3" />

    <!-- Configuration Job Bean -->
    <bean id="myJob" class="xxx.MyJob" />

    <elasticjob:job id="${myJob.id}" job-ref="myJob" registry-center-ref="regCenter
" sharding-total-count="3" cron="0/1 * * * * ?" job-listener-types=
"simpleJobListener,distributeOnceJobListener">
    </elasticjob:job>
</beans>
```

Job start

The xml that configures the Spring namespace is started through Spring, and the job will be automatically loaded.

Tracing

ElasticJob provides a tracing function, which can handle important events in the scheduling process through event subscription for query, statistics and monitor. Now, the event subscription based on relation database is provided to record events, and developers can also extend it through SPI.

Use Java API

ElasticJob-Lite currently provides `TracingConfiguration` based on database in the configuration. Developers can also extend it through SPI.

```
    // init DataSource
    DataSource dataSource = ...;
    // define tracing configuration based on relation database
    TracingConfiguration tracingConfig = new TracingConfiguration<>("RDB",
dataSource);
    // init registry center
    CoordinatorRegistryCenter regCenter = ...;
    // init job configuration
    JobConfiguration jobConfig = ...;
jobConfig.getExtraConfigurations().add(tracingConfig);
    new ScheduleJobBootstrap(regCenter, jobConfig).schedule();
```

Use Spring Boot Starter

ElasticJob-Lite Spring Boot Starter has already integrated TracingConfiguration configuration. What developers need to do is register a bean of DataSource into the Spring IoC Container and set the type of data source. Then the Starter will create an instance of TracingConfiguration and register it into the container.

Import Maven Dependency

Import spring-boot-starter-jdbc for DataSource register or create a bean of DataSource manually.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
    <version>${springboot.version}</version>
</dependency>
```

Configuration

```
spring:
  datasource:
    url: jdbc:h2:mem:job_event_storage
    driver-class-name: org.h2.Driver
    username: sa
    password:

elasticjob:
  tracing:
    type: RDB
```

Job Start

TracingConfiguration will be registered into the IoC container imperceptibly after setting tracing type to RDB. If elasticjob-lite-spring-boot-starter was imported, developers need to do nothing else. The instances of JobBootstrap will use the TracingConfiguration automatically.

Use Spring Namespace

Import Maven Dependency

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-lite-spring-namespace</artifactId>
    <version>${elasticjob.latest.version}</version>
</dependency>
```

Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:elasticjob="http://shardingsphere.apache.org/schema/elasticjob"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.
xsd
                        http://shardingsphere.apache.org/schema/elasticjob
                        http://shardingsphere.apache.org/schema/elasticjob/
elasticjob.xsd
                        ">
    <!-- Configure registry center for job -->
    <elasticjob:zookeeper id="regCenter" server-lists="yourhost:2181" namespace=
"my-job" base-sleep-time-milliseconds="1000" max-sleep-time-milliseconds="3000"
max-retries="3" />
```

```
    <!-- Configure job java bean -->
    <bean id="myJob" class="xxx.MyJob" />

    <!-- Configure DataSource -->
    <bean id="tracingDataSource" class="org.apache.commons.dbcp2.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="${driver.class.name}" />
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
    </bean>
    <!-- Configure event tracing -->
    <elasticjob:rdb-tracing id="elasticJobTrace" data-source-ref=
"elasticJobTracingDataSource" />

    <!-- Configure job -->
    <elasticjob:job id="${myJob.id}" job-ref="myJob" registry-center-ref="regCenter
" tracing-ref="elasticJobTrace" sharding-total-count="3" cron="0/1 * * * * ?" />
</beans>
```

Job Start

If the Spring container start, the `XML` that configures the Spring namespace will be loaded, and the job will be automatically started.

Table Structure

The database which is the value of the event tracing property `event_trace_rdb_url` will automatically creates two tables `JOB_EXECUTION_LOG` and `JOB_STATUS_TRACE_LOG` and several indexes.

JOB_EXECUTION_LOG Columns

Column name	Column type	Required	Describe
id	VARCHAR(40)	Yes	Primary key
job_name	VARCHAR(100)	Yes	Job name
task_id	VARCHAR(1000)	Yes	Task name, create new tasks every time the job runs.
hostname	VARCHAR(255)	Yes	Hostname
ip	VARCHAR(50)	Yes	IP
sharding_item	INT	Yes	Sharding item
execution_source	VARCHAR(20)	Yes	Source of job execution. The value options are `NORMAL_TRIGGER`, `MISFIRE`, `FAILOVER`.
failure_cause	VARCHAR(2000)	No	The reason for execution failure
is_success	BIT	Yes	Execute successfully or not
start_time	TIMESTAMP	Yes	Job start time
complete_time	TIMESTAMP	No	Job end time

`JOB_EXECUTION_LOG` records the execution history of each job. There are two steps:

1. When the job is executed, program will create one record in the `JOB_EXECUTION_LOG`, and all fields except `failure_cause` and `complete_time` are not empty.
2. When the job completes execution, program will update the record, update the columns of `is_success`, `complete_time` and `failure_cause`(if the job execution fails).

JOB_STATUS_TRACE_LOG Columns

Column name	Column type	Required	Describe
id	VARCHAR(40)	Yes	Primary key
job_name	VARCHAR(100)	Yes	Job name
original_task_id	VARCHAR(1000)	Yes	Original task name
task_id	VARCHAR(1000)	Yes	Task name
slave_id	VARCHAR(1000)	Yes	Server's name of executing the job. The valve is server's IP for `ElasticJob-Lite`, is `Mesos`' s primary key for `ElasticJob-Cloud`.
source	VARCHAR(50)	Yes	Source of job execution, the value options are `CLOUD_SCHEDULER`, `CLOUD_EXECUTOR`, `LITE_EXECUTOR`.
execution_type	VARCHAR(20)	Yes	Type of job execution, the value options are `NORMAL_TRIGGER`, `MISFIRE`, `FAILOVER`.
sharding_item	VARCHAR(255)	Yes	Collection of sharding item, multiple sharding items are separated by commas.
state	VARCHAR(20)	Yes	State of job execution, the value options are `TASK_STAGING`, `TASK_RUNNING`, `TASK_FINISHED`, `TASK_KILLED`, `TASK_LOST`, `TASK_FAILED`, `TASK_ERROR`.
message	VARCHAR(2000)	Yes	Message
creation_time	TIMESTAMP	Yes	Create time

`JOB_STATUS_TRACE_LOG` record the job status changes. Through the `task_id` of each job, user can query the life cycle and running track of the job status change.

Operation API

ElasticJob-Lite provides a Java API, which can control the life cycle of jobs in a distributed environment by directly operating the registry.

The module is still in incubation.

Configuration API

Class name: `org.apache.shardingsphere.elasticjob.lite.lifecycle.api.JobConfigurationAPI`

Get job configuration

Method signature：YamlJobConfiguration getJobConfiguration(String jobName)

- **Parameters:**
  - jobName —Job name
- **Returns:** Job configuration object

Update job configuration

Method signature：void updateJobConfiguration(YamlJobConfiguration yamlJobConfiguration)

- **Parameters:**
  - jobConfiguration —Job configuration object

Remove job configuration

Method signature：void removeJobConfiguration(String jobName)

- **Parameters:**
  - jobName —Job name

Operation API

Class name：`org.apache.shardingsphere.elasticjob.lite.lifecycle.api.JobOperateAPI`

Trigger job execution

The job will only trigger execution if it does not conflict with the currently running job, and this flag will be automatically cleared after it is started.

Method signature：void trigger(Optional jobName, Optional serverIp)

- **Parameters:**
  - jobName —Job name
  - serverIp —IP address of the job server

Disable job

Disabling a job will cause other distributed jobs to trigger resharding.

Method signature：void disable(Optional jobName, Optional serverIp)

- **Parameters:**
  - jobName —Job name
  - serverIp —job server IP address

Enable job

Method signature：void enable(Optional jobName, Optional serverIp)

- **Parameters:**
  - jobName —Job name
  - serverIp —job server IP address

Shutdown scheduling job

Method signature：void shutdown(Optional jobName, Optional serverIp)

- **Parameters:**
  - jobName —Job name
  - serverIp —IP address of the job server

Remove job

Method signature：void remove(Optional jobName, Optional serverIp)

- **Parameters:**
  - jobName —Job name
  - serverIp —IP address of the job server

Operate sharding API

Class name：`org.apache.shardingsphere.elasticjob.lite.lifecycle.api.ShardingOperateAPI`

Disable job sharding

Method signature：void disable(String jobName, String item)

- **Parameters:**
  - jobName —Job name
  - item —Job sharding item

Enable job sharding

Method signature：void enable(String jobName, String item)

- **Parameters:**
  - jobName —Job name
  - item —Job sharding item

Job statistics API

Class name：`org.apache.shardingsphere.elasticjob.lite.lifecycle.api.JobStatisticsAPI`

Get the total count of jobs

Method signature：int getJobsTotalCount()

- **Returns:** the total count of jobs

Get brief job information

Method signature：JobBriefInfo getJobBriefInfo(String jobName)

- **Parameters:**
  - jobName —Job name
- **Returns:** The brief job information

Get brief information about all jobs.

Method signature：Collection getAllJobsBriefInfo()

- **Returns:** Brief collection of all job information

Get brief information of all jobs under this IP

Method signature：Collection getJobsBriefInfo(String ip)

- **Parameters:**
  - ip —server IP
- **Returns:** Brief collection of job information

Job server status display API

Class name：`org.apache.shardingsphere.elasticjob.lite.lifecycle.api.ServerStatisticsAPI`

Total count of job servers

Method signature：int getServersTotalCount()

- **Returns:** Get the total count of job servers

Get brief information about all job servers

Method signature：Collection getAllServersBriefInfo()

- **Returns:** Brief collection of job information

Job sharding status display API

Class name: `org.apache.shardingsphere.elasticjob.lite.lifecycle.api.ShardingStatisticsAPI`

Get job sharding information collection

Method signature: Collection getShardingInfo(String jobName)

- **Parameters:**
  - jobName —Job name
- **Returns:** The collection of job sharding information

6.1.4 Configuration

Through which developers can quickly and clearly understand the functions provided by ElasticJob-Lite.

This chapter is a configuration manual for ElasticJob-Lite, which can also be referred to as a dictionary if necessary.

ElasticJob-Lite has provided 3 kinds of configuration methods for different situations.

Registry Center Configuration

Configuration

Name	Data Type	Default Value	Description
serverLists	String		ZooKeeper server IP list
namespace	String		ZooKeeper namespace
baseSleepTimeMilliseconds	int	1000	The initial value of milliseconds for the retry interval
maxSleepTimeMilliseconds	String	3000	The maximum value of milliseconds for the retry interval
maxRetries	String	3	Maximum number of retries
sessionTimeOutMilliseconds	boolean	60000	Session timeout in milliseconds
connectionTimeOutMilliseconds	boolean	15000	Connection timeout in milliseconds
digest	String	no need	Permission token to connect to ZooKeeper

Core Configuration Description

serverLists:

Include IP and port, multiple addresses are separated by commas, such as: `host1:2181,host2:2181`

Job Configuration

Configuration

Name	Data Type	Default Value	Description
jobName	St ring		Job name
sharding TotalCount	int		Sharding total count
cron	St ring		CRON expression, control the job trigger time
timeZone	St ring		time zone of CRON
sh ardingItem Parameters	St ring		Sharding item parameters
jo bParameter	St ring		Job parameter
monito rExecution	boo lean	true	Monitor job execution status
failover	boo lean	false	Enable or disable job failover
misfire	boo lean	true	Enable or disable the missed task to re-execute
maxTimeD iffSec- onds	int	-1(no check)	The maximum value for time difference between server and registry center in seconds
reco ncileInter valMinutes	int	10	Service scheduling interval in minutes for repairing job server inconsistent state
job ShardingSt rate- gyType	St ring	AVG_ALL OCATION	Job sharding strategy type
jobExecut orSer- viceH andlerType	St ring	CPU	Job thread pool handler type
jobErrorH andler- Type	St ring		Job error handler type
d escription	St ring		Job description
props	Pr oper ties		Job properties
disabled	boo lean	false	Enable or disable start the job
overwrite	boo lean	false	Enable or disable local configuration override registry center configuration

Core Configuration Description

shardingItemParameters:

The sequence numbers and parameters of the Sharding items are separated by equal sign, and multiple key-value pairs are separated by commas. The Sharding sequence number starts from `0` and can't be greater than or equal to the total number of job fragments. For example: `0=a,1=b,2=c`

jobParameter:

With this parameter, user can pass parameters for the business method of job scheduling, which is used to implement the job with parameters. For example: `Amount of data acquired each time`, `Primary key of the job instance read from the database`, etc.

monitorExecution:

When the execution time and interval of each job are very short, it is recommended not to monitor the running status of the job to improve efficiency. There is no need to monitor because it is a transient state. User can add data accumulation monitoring by self. And there is no guarantee that the data will be selected repeatedly, idempotency should be achieved in the job. If the job execution time and interval time are longer, it is recommended to monitor the job status, and it can guarantee that the data will not be selected repeatedly.

maxTimeDiffSeconds:

If the time error exceeds the configured seconds, an exception will be thrown when the job starts.

reconcileIntervalMinutes:

In a distributed system, due to network, clock and other reasons, ZooKeeper may be inconsistent with the actual running job. This inconsistency cannot be completely avoided through positive verification. It is necessary to start another thread to periodically calibrate the consistency between the registry center and the job status, that is, to maintain the final consistency of ElasticJob.

Less than `1` means no repair is performed.

jobShardingStrategyType:

For details, seeJob Sharding Strategy。

jobExecutorServiceHandlerType:

For details, seeThread Pool Strategy。

jobErrorHandlerType:

For details, seeError Handler Strategy。

props:

For details, seeJob Properties。

disabled:

It can be used for deployment, forbid jobs to start, and then start them uniformly after the deployment is completed.

overwrite:

If the value is `true`, local configuration override registry center configuration every time the job is started.

Job Listener Configuration

Common Listener Configuration

Configuration: no

Distributed Listener Configuration

Configuration

Name	Data Type	D efault Value	Description
started-timeo ut-milliseconds	long	Lo ng.MAX _VALUE	The timeout in milliseconds before the last job is executed
completed-timeo ut-milliseconds	long	Lo ng.MAX _VALUE	The timeout in milliseconds after the last job is executed

Event Tracing Configuration

Configuration

Name	Data Type	Default Value	Description
type	String		The type of event tracing storage adapter
storage	Generics Type		The object of event tracing storage adapter

Java API

Registry Center Configuration

The component which is used to register and coordinate the distributed behavior of jobs, currently only supports `ZooKeeper`.

Class name: `org.apache.shardingsphere.elasticjob.reg.zookeeper.ZookeeperConfiguration`

Configuration:

Name	Constructor injection
serverLists	Yes
namespace	Yes
baseSleepTimeMilliseconds	No
maxSleepTimeMilliseconds	No
maxRetries	No
sessionTimeoutMilliseconds	No
connectionTimeoutMilliseconds	No
digest	No

Job Configuration

Class name: `org.apache.shardingsphere.elasticjob.api.JobConfiguration`

Configuration:

Name	Constructor injection
jobName	Yes
shardingTotalCount	Yes
cron	No
timeZone	No
shardingItemParameters	No
jobParameter	No
monitorExecution	No
failover	No
misfire	No
maxTimeDiffSeconds	No
reconcileIntervalMinutes	No
jobShardingStrategyType	No
jobExecutorServiceHandlerType	No
jobErrorHandlerType	No
jobListenerTypes	No
description	No
props	No
disabled	No
overwrite	No

Spring Boot Starter

To use the Spring boot, user need to add the dependency of the `elasticjob-lite-spring-boot-starter` module in the `pom.xml` file.

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-lite-spring-boot-starter</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Registry Center Configuration

Prefix: `elasticjob.reg-center`

Configuration:

Property name	Required
server-lists	Yes
namespace	Yes
base-sleep-time-milliseconds	No
max-sleep-time-milliseconds	No
max-retries	No
session-timeout-milliseconds	No
connection-timeout-milliseconds	No
digest	No

Reference:

YAML

```
elasticjob:
  regCenter:
    serverLists: localhost:6181
    namespace: elasticjob-lite-springboot
```

Properties

```
elasticjob.reg-center.namespace=elasticjob-lite-springboot
elasticjob.reg-center.server-lists=localhost:6181
```

Job Configuration

Prefix: `elasticjob.jobs`

Configuration:

Property name	Required
elasticJobClass / elasticJobType	Yes
cron	No
timeZone	No
jobBootstrapBeanName	No
sharding-total-count	Yes
sharding-item-parameters	No
job-parameter	No
monitor-execution	No
failover	No
misfire	No
max-time-diff-seconds	No
reconcile-interval-minutes	No
job-sharding-strategy-type	No
job-executor-service-handler-type	No
job-error-handler-type	No
job-listener-types	No
description	No
props	No
disabled	No
overwrite	No

“elasticJobClass” and “elasticJobType” are mutually exclusive.

If cron was configured, the job will be created as a ScheduleJobBootstrap. The Starter will start scheduling when application is ready. Otherwise, the job will be created as a OneOffJobBootstrap with a name specified by “jobBootstrapBeanName”. It requires manual injection and execution.

Reference:

YAML

```
elasticjob:
  jobs:
    simpleJob:
      elasticJobClass: org.apache.shardingsphere.elasticjob.lite.example.job.
SpringBootSimpleJob
      cron: 0/5 * * * * ?
      timeZone: GMT+08:00
      shardingTotalCount: 3
      shardingItemParameters: 0=Beijing,1=Shanghai,2=Guangzhou
```

```
  scriptJob:
    elasticJobType: SCRIPT
    cron: 0/10 * * * * ?
    shardingTotalCount: 3
    props:
      script.command.line: "echo SCRIPT Job: "
  manualScriptJob:
    elasticJobType: SCRIPT
    jobBootstrapBeanName: manualScriptJobBean
    shardingTotalCount: 9
    props:
      script.command.line: "echo Manual SCRIPT Job: "
```

Properties

```
elasticjob.jobs.simpleJob.elastic-job-class=org.apache.shardingsphere.elasticjob.
lite.example.job.SpringBootSimpleJob
elasticjob.jobs.simpleJob.cron=0/5 * * * * ?
elasticjob.jobs.simpleJob.timeZone=GMT+08:00
elasticjob.jobs.simpleJob.sharding-total-count=3
elasticjob.jobs.simpleJob.sharding-item-parameters=0=Beijing,1=Shanghai,2=Guangzhou
elasticjob.jobs.scriptJob.elastic-job-type=SCRIPT
elasticjob.jobs.scriptJob.cron=0/5 * * * * ?
elasticjob.jobs.scriptJob.sharding-total-count=3
elasticjob.jobs.scriptJob.props.script.command.line=echo SCRIPT Job:
elasticjob.jobs.manualScriptJob.elastic-job-type=SCRIPT
elasticjob.jobs.manualScriptJob.job-bootstrap-bean-name=manualScriptJobBean
elasticjob.jobs.manualScriptJob.sharding-total-count=3
elasticjob.jobs.manualScriptJob.props.script.command.line=echo Manual SCRIPT Job:
```

Event Trace Configuration

Prefix: `elasticjob.tracing`

Property name	Options	Required	Description
type	RDB	No	
includeJobNames		No	allow list of job
excludeJobNames		No	block list of job

“includeJobNames” and “excludeJobNames” are mutually exclusive.

Load all Job When “includeJobNames” and “excludeJobNames” are null.

RDB is the only supported type at present. If Spring IoC container contained a bean of DataSource and RDB was set in configuration, an instance of TracingConfiguration will be created automatically.

Reference:

YAML

```
elasticjob:
  tracing:
    type: RDB
    excludeJobNames: [ job-name-1, job-name-2 ]
```

Properties

```
elasticjob.tracing.type=RDB
elasticjob.tracing.excludeJobNames=[ job-name ]
```

Dump Job Info Configuration

Prefix: `elasticjob.dump`

Property name	Default value	Required
enabled	true	No
port		Yes

Designate a port as dump port in configurations. The Spring Boot Starter will enable dumping automatically. If the port for job dump was missing, dump won't be enabled.

Reference:

YAML

```
elasticjob:
  dump:
    port: 9888
```

Properties

```
elasticjob.dump.port=9888
```

Spring Namespace

To use the Spring namespace, user need to add the dependency of the `elasticjob-lite-spring` module in the `pom.xml` file.

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-lite-spring-namespace</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Spring namespace: http://shardingsphere.apache.org/schema/elasticjob/elasticjob.xsd

Registry Center Configuration

<elasticjob:zookeeper />

Configuration:

Name	Required
id	Yes
server-lists	Yes
namespace	Yes
base-sleep-time-milliseconds	No
max-sleep-time-milliseconds	No
max-retries	No
session-timeout-milliseconds	No
connection-timeout-milliseconds	No
digest	No

Job Configuration

<elasticjob:job />

Configuration:

Name	Required
id	Yes
class	No
job-ref	No
registry-center-ref	Yes
tracing-ref	No
cron	Yes
timeZone	No
sharding-total-count	Yes
sharding-item-parameters	No
job-parameter	No
monitor-execution	No
failover	No
misfire	No
max-time-diff-seconds	No
reconcile-interval-minutes	No
job-sharding-strategy-type	No
job-executor-service-handler-type	No
job-error-handler-type	No
description	No
props	No
disabled	No
overwrite	No

Event Tracing Configuration

<elasticjob:rdb-tracing />

Configuration:

Name	Data Type	Required	Default Value	Description
id	String	Yes		The bean's identify of the event tracing
data-source-ref	DataSource	No		The bean's name of the event tracing DataSource

Job Dump Configuration

<elasticjob:snapshot />

Configuration:

Name	D ata T ype	R e q u i r e d	Def ault V alue	Description
id	Str ing	Y e s		The identify of the monitoring service in the Spring container
regis try-cen ter-ref	Str ing	Y e s		Registry center bean’ s reference, need to the statement of the `reg:zookeeper`
du mp-port	Str ing	Y e s		Job dump portusage: echo “dump@jobName” | nc 127.0.0.1 9888

Built-in Strategy

Introduction

ElasticJob allows developers to implement strategies via SPI; At the same time, ElasticJob also provides a couple of built-in strategies for simplify developers.

Usage

The built-in strategies are configured by type. This chapter distinguishes and lists all the built-in strategies of ElasticJob according to its functions for developers’ reference.

Job Sharding Strategy

Average Allocation Strategy

Type: AVG_ALLOCATION

Sharding or average by sharding item.

If the job server number and sharding count cannot be divided, the redundant sharding item that cannot be divided will be added to the server with small sequence number in turn.

For example: 1. If there are 3 job servers and the total sharding count is 9, each job server is divided into: 1=[0,1,2], 2=[3,4,5], 3=[6,7,8]; 2. If there are 3 job servers and the total sharding count is 8, each job server is divided into: 1=[0,1,6], 2=[2,3,7], 3=[4,5]; 3. If there are 3 job servers and the total sharding count is 10, each job server is divided into: 1=[0,1,2,9], 2=[3,4,5], 3=[6,7,8].

Odevity Strategy

Type: ODEVITY

Sharding for hash with job name to determine IP asc or desc.

IP address asc if job name’ hashcode is odd; IP address desc if job name’ hashcode is even. Used to average assign to job server.

For example: 1. If there are 3 job servers with 2 sharding item, and the hash value of job name is odd, then each server is divided into: 1 = [0], 2 = [1], 3 = []; 2. If there are 3 job servers with 2 sharding item, and the hash value of job name is even, then each server is divided into: 3 = [0], 2 = [1], 1 = [].

Round Robin Strategy

Type: ROUND_ROBIN

Sharding for round robin by name job.

Thread Pool Strategy

CPU Resource Strategy

Type: CPU

Use CPU available processors * 2 to create thread pool.

Single Thread Strategy

Type: SINGLE_THREAD

Use single thread to execute job.

Error Handler Strategy

Log Strategy

Type: LOG

Built-in: Yes

Log error and do not interrupt job.

Throw Strategy

Type: THROW

Built-in: Yes

Throw system exception and interrupt job.

Ignore Strategy

Type: IGNORE

Built-in: Yes

Ignore exception and do not interrupt job.

Email Notification Strategy

Type: EMAIL

Built-in: No

Send email message notification and do not interrupt job.

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-email</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Configuration:

Name	Description	Required	Default Value
email.host	Email server host address	Yes	.
email.port	Email server port	Yes	.
email.username	Email server username	Yes	.
email.password	Email server password	Yes	.
email.useSsl	Whether to enable SSL encrypted transmission	No	true
email.subject	Email Subject	No	ElasticJob error message
email.from	Sender email address	Yes	.
email.to	Recipient's email address	Yes	.
email.cc	Carbon copy email address	No	null
email.bcc	Blind carbon copy email address	No	null
email.debug	Whether to enable debug mode	No	false

Wechat Enterprise Notification Strategy

Type: WECHAT

Built-in: No

Send wechat message notification and do not interrupt job

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-wechat</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Configuration:

Name	Description	Required	Default Value
wechat.webhook	The webhook address of the wechat robot	Yes	•
wechat.connectTimeoutMilliseconds	The timeout period for establishing a connection with the wechat server	No	3000 milliseconds
wechat.readTimeoutMilliseconds	The timeout period for reading available resources from the wechat server	No	5000 milliseconds

Dingtalk Notification Strategy

Type: DINGTALK

Built-in: No

Send dingtalk message notification and do not interrupt job

Maven POM:

```
<dependency>
    <groupId>org.apache.shardingsphere.elasticjob</groupId>
    <artifactId>elasticjob-error-handler-dingtalk</artifactId>
    <version>${latest.release.version}</version>
</dependency>
```

Configuration:

Name	Description	Required	Default Value
dingtalk.webhook	The webhook address of the dingtalk robot	Yes	•
dingtalk.keyword	Custom keywords	No	null
dingtalk.secret	Secret for dingtalk robot	No	null
dingtalk.connectTimeoutMilliseconds	The timeout period for establishing a connection with the dingtalk server	No	3000 milliseconds
dingtalk.readTimeoutMilliseconds	The timeout period for reading available resources from the dingtalk server	No	5000 milliseconds

Job Properties

Introduction

`ElasticJob` provide customized configurations for different types of jobs through the way of attribute configuration.

Job Type

Simple Job

Interface name: `org.apache.shardingsphere.elasticjob.simple.job.SimpleJob`

Configuration: no

Dataflow Job

Interface name: `org.apache.shardingsphere.elasticjob.dataflow.job.DataflowJob`

Configuration:

Name	*Data Type*	*Description*	*Default Value*
streaming.process	boolean	Enable or disable Streaming	false

Script Job

Type: `SCRIPT`

Configuration:

Name	*Data Type*	*Description*	*Default Value*
script.command.line	String	Script content or path	.

HTTP Job

Type：`HTTP`

Configuration:

Name	*Data Type*	*Description*	*Default Value*
http.url	String	http request url	.
http.method	String	http request method	.
http.data	String	http request data	.
htt p.connect.timeo ut.milliseconds	String	http connect timeout	3000
http.read.timeo ut.milliseconds	String	http read timeout	5000
ht tp.content.type	String	http content type	.

6.1.5 Operation

This chapter is an operation manual for ElasticJob-Lite.

Deploy Guide

Application deployment

1. Start the ZooKeeper of the ElasticJob-Lite designated registry.
2. Run the jar file containing ElasticJob-Lite and business code. It is not limited to the startup mode of jar or war.
3. When the job server is configured with multiple network cards, the network card address can be specified by setting the system variable `elasticjob.preferred.network.interface` or specify network addresses by setting the system variable `elasticjob.preferred.network.ip`. ElasticJob obtains the first non-loopback available IPV4 address in the network card list by default.

Operation and maintenance platform and RESTFul API deployment (optional)

1. Unzip `elasticjob-lite-console-${version}.tar.gz` and execute `bin\start.sh`.
2. Open the browser and visit `http://localhost:8899/` to access the console. 8899 is the default port number. You can customize the port number by entering `-p` through the startup script.
3. The method of accessing RESTFul API is the same as the console.
4. `elasticjob-lite-console-${version}.tar.gz` can be obtained by compiling `mvn install`.

Dump Job Information

Using ElasticJob may meet some distributed problem which is not easy to observe.

Because of developer cannot debug in production environment, ElasticJob provide dump command to export job runtime information for debugging.

For security reason, the information dumped had already mask sensitive information, it instead of real IP address to `ip1`, `ip2` ⋯

Open Listener Port

Using Java API please refer to Java API usage for more details. Using Spring please refer to Spring usage for more details.

Execute Dump

Dump to stdout

```
echo "dump@jobName" | nc <job server IP address> 9888
```

```
[chris:elastic-job]echo "dump" | nc localhost 9888
/simpleElasticJob/servers |
/simpleElasticJob/servers/ip1 |
/simpleElasticJob/servers/ip1/status | READY
/simpleElasticJob/servers/ip1/sharding | 0,1,2,3,4,5,6,7,8,9
/simpleElasticJob/servers/ip1/hostName | localhost
/simpleElasticJob/leader |
/simpleElasticJob/leader/sharding |
/simpleElasticJob/leader/execution |
/simpleElasticJob/leader/election |
/simpleElasticJob/leader/election/latch |
/simpleElasticJob/leader/election/host | ip1
/simpleElasticJob/config |
/simpleElasticJob/config/shardingTotalCount | 10
/simpleElasticJob/config/shardingItemParameters | 0=A,1=B,2=C,3=D,4=E,5=F,6=G,7=H,8=I,9=J
/simpleElasticJob/config/processCountIntervalSeconds | 300
/simpleElasticJob/config/monitorPort | 9888
/simpleElasticJob/config/monitorExecution | false
/simpleElasticJob/config/misfire | true
/simpleElasticJob/config/maxTimeDiffSeconds | -1
/simpleElasticJob/config/jobShardingStrategyClass |
/simpleElasticJob/config/jobParameter |
/simpleElasticJob/config/jobClass | com.dangdang.example.elasticjob.spring.job.SimpleJobDemo
/simpleElasticJob/config/fetchDataCount | 1
/simpleElasticJob/config/failover | true
/simpleElasticJob/config/description | 只运行一次的作业示例
/simpleElasticJob/config/cron | 0/5 * * * * ?
/simpleElasticJob/config/concurrentDataProcessThreadCount | 1
```

Figure2: Dump

Dump to file

```
echo "dump@jobName" | nc <job server IP address> 9888 > job_debug.txt
```

Execution Monitor

By monitoring several key nodes in the zookeeper registry of ElasticJob-Lite, the job running status monitoring function can be completed.

Monitoring job server alive

Listen for the existence of node job_name:raw-latex:*instances*:raw-latex:`\job`_instance_id. This node is a temporary node. If the job server is offline, the node will be deleted.

Console

Unzip `elasticjob-lite-console-${version}.tar.gz` and execute `bin\start.sh`. Open the browser and visit `http://localhost:8899/` to access the console. 8899 is the default port number. You can customize the port number by entering `-p` through the startup script.

Log in

The console provides two types of accounts: administrator and guest. The administrator has all operation rights, and the visitors only have the viewing rights. The default administrator user name and password are root/root, and the guest user name and password are guest/guest, You can modify the administrator and guest user names and passwords through `conf\application.properties`.

```
auth.root_username=root
auth.root_password=root
auth.guest_username=guest
auth.guest_password=guest
```

Function list

- Login security control
- Registration center, event tracking data source management
- Quickly modify job settings
- View job and server dimension status
- Operational job disable/enable, stop and delete life cycle
- Event tracking query

Design concept

The operation and maintenance platform has no direct relationship with ElasticJob-Lite. It displays the job status by reading the job registration center data, or updating the registration center data to modify the global configuration.

The console can only control whether the job itself is running, but it cannot control the start of the job process, because the console and the job server are completely separated, and the console cannot control the job server.

Unsupported item

- Add assignment

The job will be automatically added the first time it runs. ElasticJob-Lite is started as a jar and has no job distribution function. To publish jobs entirely through the operation and maintenance platform, please use ElasticJob-Cloud.

6.2 ElasticJob-Cloud

6.2.1 Introduction

ElasticJob-Cloud uses Mesos to manage and isolate resources.

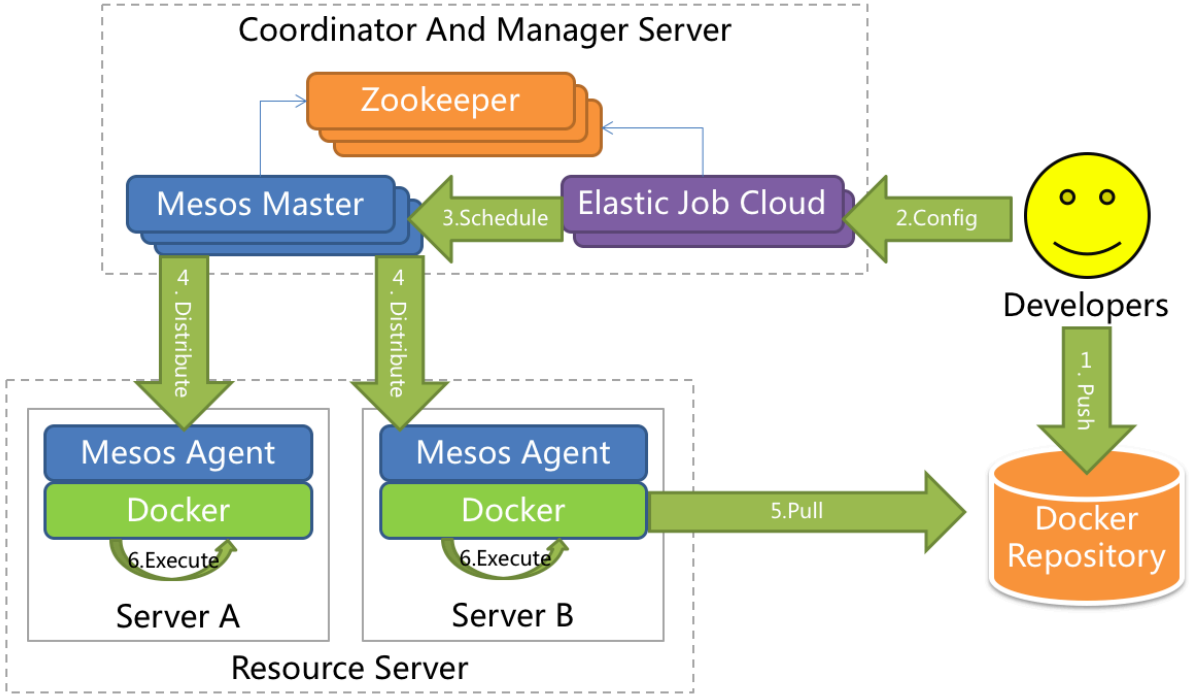

Figure3: ElasticJob-Cloud Architecture

6.2.2 Comparison

	ElasticJob-Lite	*ElasticJob-Cloud*
Decentralization	Yes	`No`
Resource Assign	No	`Yes`
Job Execution	Daemon	`Daemon + Transient`
Deploy Dependency	ZooKeeper	`ZooKeeper + Mesos`

The advantages of ElasticJob-Cloud are resource management and isolation, which is suitable for big data application with starve resource environment.

6.2.3 Usage

This chapter will introduce the use of ElasticJob-Cloud. Please refer to Example for more details.

Dev Guide

Job development

ElasticJob-Lite and ElasticJob-Cloud provide a unified job interface, developers only need to develop business jobs once, and then they can deploy to different environments according to different configurations.

For details of job development, please refer to ElasticJob-Lite user manual.

Job start

You need to define the `main` method and call it `JobBootstrap.execute()`, for example:

```
public class MyJobDemo {

    public static void main(final String[] args) {
        JobBootstrap.execute(new MyJob());
    }
}
```

Local Executor

When developing ElasticJob-Cloud jobs, developers can leave the Mesos environment to run and debug jobs locally. The local operating mode can be used to fully debug business functions and unit tests, and then deploy to the Mesos cluster after completion.

There is no need to install the Mesos environment to run jobs locally.

```
// Create job configuration
JobConfiguration jobConfig = JobConfiguration.newBuilder("myJob", 3).cron("0/5 * *
* * * ?").build();

// Configure the fragmentation item of the currently running job
int shardingItem = 0;

// Create a local executor
new LocalTaskExecutor(new MyJob(), jobConfig, shardingItem).execute();
```

6.2.4 Configuration

ElasticJob-Cloud provides RESTful APIs such as application publishing and job registration, which can be operated by curl.

Request URL prefix is `/api`

Authentication API

Get AccessToken

url: login

Method: POST

Content type: application/json

Parameter list:

Property name	Type	Required or not	Default value	Description
username	String	Yes		API au thentication username
password	String	Yes		API au thentication password

Response parameter:

Property name	Type	Description
accessToken	String	API authentication token

Example:

```
curl -H "Content-Type: application/json" -X POST http://elasticjob_cloud_host:8899/
api/login -d '{"username": "root", "password": "pwd"}'
```

Response body:

```
{"accessToken":"some_token"}
```

Application API

Publish application

url: app

Method: POST

Parameter type: application/json

Parameter list:

Property name	Type	Required or not	Default value	Description
appName	String	Yes		Job application name
appURL	String	Yes		Path of job application
cpuCount	double	No	1	The number of CPUs required for the job application to start
memoryMB	double	No	128	MB of memory required to start the job application
boots trapScript	String	Yes		Boot script
appCacheEnable	boolean	No	true	Whether to read the application from the cache every time the job is executed
eventTraceSamplingCount	int	No	0 (no sampling)	Number of resident job event sampling rate statistics

Detailed parameter description:

appName:

It is the unique identifier of ElasticJob-Cloud's job application.

appURL:

A path that can be accessed through the network must be provided.

bootstrapScript:

Example: bin:raw-latex:*start*.sh

appCacheEnable:

Disabled, every time the task is executed, the application will be downloaded from the application repository to the local.

eventTraceSamplingCount:

To avoid excessive data volume, you can configure the sampling rate for frequently scheduled resident jobs, that is, every N times the job is executed, the job execution and tracking related data will be recorded.

Example:

```
curl -l -H "Content-type: application/json" -X POST -d '{"appName":"my_app","appURL
":"http://app_host:8080/my-job.tar.gz","cpuCount":0.1,"memoryMB":64.0,
"bootstrapScript":"bin/start.sh","appCacheEnable":true,"eventTraceSamplingCount":0}
' http://elastic_job_cloud_host:8899/api/app
```

Modify application configuration

url: app

Method: PUT

Parameter type: application/json

Parameter list:

Property name	Type	Required or not	Default value	Description
appName	String	Yes		Job application name
appCacheEnable	boolean	Yes	true	Whether to read the application from the cache every time the job is executed
eventTraceSamplingCount	int	No	0 (no sampling)	Number of resident job event sampling rate statistics

Example:

```
curl -l -H "Content-type: application/json" -X PUT -d '{"appName":"my_app",
"appCacheEnable":true}' http://elastic_job_cloud_host:8899/api/app
```

Job API

Register job

url: job/register

Method: POST

Parameter type: application/json

Parameter list:

Property name	Type	Required or not	Default value	Description
appName	String	Yes		Job application name
cpuCount	double	Yes		The number of CPUs required for a single chip operation, the minimum value is 0.001
memoryMB	double	Yes		The memory MB required for a single chip operation, the minimum is 1
jobExecutionType	Enum	Yes		Job execution type. TRANSIENT is a transient operation, DAEMON is a resident operation
jobName	String	Yes		Job name
cron	String	No		cron expression, used to configure job trigger time
shardingTotalCount	int	Yes		Total number of job shards
shardingItemParameters	String	No		Custom sharding parameters
jobParameter	String	No		Job custom parameters
failover	boolean	No	false	Whether to enable failover
misfire	boolean	No	false	Whether to enable missed tasks to re-execute
jobExecutorServiceHandlerType	boolean	No	false	Job thread pool processing strategy
jobErrorHandlerType	boolean	No	false	Job error handling strategy
description	String	No		Job description information
props	Properties	No		Job property configuration information

Use the script type instantaneous job to upload the script directly to appURL without tar package. If there is only a single script file, no compression is required. If it is a complex script application, you can still upload a tar package and support various common compression formats.

Example:

```
curl -l -H "Content-type: application/json" -X POST -d '{"appName":"my_app",
"cpuCount":0.1,"memoryMB":64.0,"jobExecutionType":"TRANSIENT","jobName":"my_job",
"cron":"0/5 * * * * ?","shardingTotalCount":5,"failover":true,"misfire":true}'
http://elastic_job_cloud_host:8899/api/job/register
```

update job configuration

url: job/update

Method: PUT

Parameter type: application/json

Parameters: same as registration job

Example:

```
curl -l -H "Content-type: application/json" -X PUT -d '{"appName":"my_app","jobName
":"my_job","cpuCount":0.1,"memoryMB":64.0,"jobExecutionType":"TRANSIENT","cron":"0/
5 * * * * ?","shardingTotalCount":5,"failover":true,"misfire":true}' http://
elastic_job_cloud_host:8899/api/job/update
```

Deregister Job

url: job/deregister

Method: DELETE

Parameter type: application/json

Parameters: Job name

Example:

```
curl -l -H "Content-type: application/json" -X DELETE -d 'my_job' http://elastic_
job_cloud_host:8899/api/job/deregister
```

Trigger job

url: job/trigger

Method: POST

Parameter type: application/json

Parameters: Job name

Description: Event-driven, triggering jobs by calling API instead of timing. Currently only valid for transient operations.

Example:

```
curl -l -H "Content-type: application/json" -X POST -d 'my_job' http://elastic_job_
cloud_host:8899/api/job/trigger
```

6.2.5 Operation

This chapter is an operation manual for ElasticJob-Cloud.

Deploy Guide

Scheduler deployment steps

1. Start ElasticJob-Cloud-Scheduler and Mesos, and specify ZooKeeper as the registry.
2. Start Mesos Master and Mesos Agent.
3. Unzip `elasticjob-cloud-scheduler-${version}.tar.gz`.
4. Run `bin\start.sh` to start ElasticJob-Cloud-Scheduler.

Job deployment steps

1. Ensure that ZooKeeper, Mesos Master/Agent and ElasticJob-Cloud-Scheduler have been started correctly.
2. Place the tar.gz file of the packaging job in a network accessible location, such as ftp or http. The `main` method in the packaged tar.gz file needs to call the `JobBootstrap.execute` method provided by ElasticJob-Cloud.
3. Use curl command to call RESTful API to publish applications and register jobs. For details: Configuration

Scheduler configuration steps

Modify the `conf\elasticjob-cloud-scheduler.properties` to change the system configuration.

Configuration description:

Attribute Name	Required	Default	Description
hostname	yes		The real IP or hostname of the server, cannot be 127.0.0.1 or localhost
user	no		User name used by Mesos framework
mesos_url	yes	zk://127.0.0.1:2181/mesos	Zookeeper url used by Mesos
zk_servers	yes	127.0.0.1:2181	Zookeeper address used by ElasticJob-Cloud
zk_namespace	no	elastic-job-cloud	Zookeeper namespace used by ElasticJob-Cloud
zk_digest	no		Zookeeper digest used by ElasticJob-Cloud
http_port	yes	8899	Port used by RESTful API
job_state_queue_size	yes	10000	The maximum value of the accumulation job, the accumulation job exceeding this threshold will be discarded. Too large value may cause ZooKeeper to become unresponsive, and should be adjusted according to the actual measurement
event_trace_rdb_driver	no		Driver of Job event tracking database
event_trace_rdb_url	no		Url of Job event tracking database
event_trace**_rdb_**username	no		Username of Job event tracking database
event_trace**_rdb_**password	no		Password of Job event tracking database
auth_username	no	root	API authentication username
auth_password	no	pwd	API authentication password

- Stop: No stop script is provided, you can directly use the kill command to terminate the process.

High Available

Introduction

The high availability of the scheduler is achieved by running several ElasticJob-Cloud-Scheduler instances pointing to the same ZooKeeper cluster. ZooKeeper is used to perform leader election when the current primary ElasticJob-Cloud-Scheduler instance fails. At least two scheduler instances are used to form a cluster. Only one scheduler instance in the cluster provides services, and the other instances are in the `standby` state. When the instance fails, the cluster will elect one of the remaining instances to continue providing services.

Configuration

Each ElasticJob-Cloud-Scheduler instance must use the same ZooKeeper cluster. For example, if the Quorum of ZooKeeper is zk://1.2.3.4:2181,2.3.4.5:2181,3.4.5.6:2181/elasticjob-cloud, the ZooKeeper related configuration in `elasticjob-cloud-scheduler.properties` is:

```
# ElasticJob-Cloud's ZooKeeper address
zk_servers=1.2.3.4:2181,2.3.4.5:2181,3.4.5.6:2181

# ElasticJob-Cloud's ZooKeeper namespace
zk_namespace=elasticjob-cloud
```

Console

The operation and maintenance platform is embedded in the jar package of elasticjob-cloud-scheduler, and there is no need to start an additional WEB server. The startup port can be adjusted by modifying the http_port parameter in the configuration file. The default port is 8899 and the access address is `http://{your_scheduler_ip}:8899`.

Log in

Two types of accounts are provided, administrator and guest. The administrator has all operation permissions, and the visitor only has viewing permissions. The default administrator user name and password are root/root, and the guest user name and password are guest/guest. You can modify the administrator and guest user names and passwords through `conf\auth.properties`.

Function list

- Application management (publish, modify, view)
- Job management (register, modify, view and delete)
- View job status (waiting to run, running, pending failover)
- Job history view (running track, execution status, historical dashboard)

Design concept

The operation and maintenance platform uses pure static HTML + JavaScript to interact with the back-end RESTful API. It displays the job configuration and status by reading the job registry, the database displays the job running track and execution status, or updates the job registry data to modify the job configuration.

7 Dev Manual

ElasticJob provides dozens of SPI based extensions. it is very convenient to customize the functions for developers.

This chapter lists all SPI extensions of ElasticJob. If there is no special requirement, users can use the built-in implementation provided by ElasticJob; advanced users can refer to the interfaces for customized implementation.

ElasticJob community welcomes developers to feed back their implementations to the open-source community, so that more users can benefit from it.

7.1 Job Sharding Strategy

Job Sharding Strategy, used to sharding job to distributed tasks.

SPI Name	*Description*
JobShardingStrategy	Job sharding strategy

Implementation Class	*Description*
AverageAlloc ationJobShardingStrategy	Sharding or average by sharding item
OdevitySortB yNameJobShardingStrategy	Sharding for hash with job name to determine IP asc or desc
RotateServerB yNameJobShardingStrategy	Sharding for round robin by name job

7.2 Thread Pool Strategy

Thread pool strategy, used to create thread pool for job execution.

SPI Name	*Description*
JobExecutorServiceHandler	Job executor service handler

Implementation Class	*Description*
CPUUsag eJobExecutorServiceHandler	Use CPU available processors * 2 to create thread pool
SingleThrea dJobExecutorServiceHandler	Use single thread to execute job

7.3 Error Handler

Error handler strategy, used to handle error when exception occur during job execution.

SPI Name	*Description*
JobErrorHandler	Job error handler

Implementation Class	*Description*
L ogJobErrorHandler	Log error and do not interrupt job
Thr owJobErrorHandler	Throw system exception and interrupt job
Igno reJobErrorHandler	Ignore exception and do not interrupt job
Ema ilJobErrorHandler	Send email message notification and do not interrupt job
Wech atJobErrorHandler	Send wechat message notification and do not interrupt job
Dingta lkJobErrorHandler	Send dingtalk message notification and do not interrupt job

7.4 Job Class Name Provider

Job class name provider, used to provide job class name in different contain environments.

SPI Name	*Description*
JobClassNameProvider	Job class name provider

Implementation Class	*Description*
Defau ltJobClassNameProvider	Job class name provider in standard environment
SpringPro xyJobClassNameProvider	Job class name provider in Spring container environment

7.5 Roadmap

7.5.1 Kernel

- ☒ Unified Job Config API
  - ☒ Core Config
  - ☒ Type Config
  - ☒ Root Config
- ☒ Job Types
  - ☒ Simple
  - ☒ Dataflow
  - ☒ Script
  - ☒ Http (Since 3.0.0-beta)
- ☒ Event Trace
  - ☒ Event Publisher
  - ☒ Database Event Listener
  - ☐ Other Event Listener
- ☐ Unified Schedule API
- ☐ Unified Resource API

7.5.2 ElasticJob-Lite

- ☒ Distributed Features
  - ☒ High Availability
  - ☒ Elastic scale in/out
  - ☒ Failover
  - ☒ Misfire
  - ☒ Idempotency
  - ☒ Reconcile
- ☒ Registry Center
  - ☒ ZooKeeper
  - ☐ Other Registry Center Supported
- ☒ Lifecycle Management
  - ☒ Add/Remove

  - ☑ Pause/Resume
  - ☑ Disable/Enable
  - ☑ Shutdown
  - ☑ Restful API
  - ☑ Web Console
- ☑ Job Dependency
  - ☑ Listener
  - ☐ DAG
- ☑ Spring Integrate
  - ☑ Namespace
  - ☑ Bean Injection
  - ☑ Spring Boot Starter (Since 3.0.0-alpha)

7.5.3 ElasticJob-Cloud

- ☑ Transient Job
  - ☑ High Availability
  - ☑ Elastic scale in/out
  - ☑ Failover
  - ☑ Misfire
  - ☑ Idempotency
- ☑ Daemon Job
  - ☑ High Availability
  - ☑ Elastic scale in/out
  - ☐ Failover
  - ☐ Misfire
  - ☑ Idempotency
- ☑ Mesos Scheduler
  - ☑ High Availability
  - ☑ Reconcile
  - ☐ Redis Based Queue Improvement
  - ☐ Http Driver
- ☑ Mesos Executor

  - ☒ Executor Reuse Pool
  - ☐ Progress Reporting
  - ☐ Health Detection
  - ☐ Log Redirect
- ☒ Lifecycle Management
  - ☒ Job Add/Remove
  - ☐ Job Pause/Resume
  - ☒ Job Disable/Enable
  - ☐ Job Shutdown
  - ☒ App Add/Remove
  - ☒ App Disable/Enable
  - ☒ Restful API
  - ☒ Web Console
- ☐ Job Dependency
  - ☐ Listener
  - ☐ Workflow
  - ☐ DAG
- ☒ Job Distribution
  - ☒ Mesos Based Distribution
  - ☐ Docker Based Distribution
- ☒ Resources Management
  - ☒ Resources Allocate
  - ☐ Cross Data Center
  - ☐ A/B Test
- ☒ Spring Integrate
  - ☒ Bean Injection

8 Downloads

8.1 Latest Releases

ElasticJob is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512.

8.1.1 ElasticJob - Version: 3.0.1 (Release Date: Oct 11, 2021)

- Source Codes: [SRC] [ASC] [SHA512]
- ElasticJob-Lite Binary Distribution: [TAR] [ASC] [SHA512]
- ElasticJob-Cloud-Scheduler Binary Distribution: [TAR] [ASC] [SHA512]
- ElasticJob-Cloud-Executor Binary Distribution: [TAR] [ASC] [SHA512]

8.1.2 ElasticJob-UI - Version: 3.0.1 (Release Date: Jan 19, 2022)

- Source Codes: [SRC] [ASC] [SHA512]
- ElasticJob-Lite-UI Binary Distribution: [TAR] [ASC] [SHA512]
- ElasticJob-Cloud-UI Binary Distribution: [TAR] [ASC] [SHA512]

8.2 All Releases

Find all releases in the Archive repository.

8.3 Verify the Releases

PGP signatures KEYS

It is essential that you verify the integrity of the downloaded files using the PGP or SHA signatures. The PGP signatures can be verified using GPG or PGP. Please download the KEYS as well as the asc signature files for relevant distribution. It is recommended to get these files from the main distribution directory and not from the mirrors.

```
gpg -i KEYS
```

or

```
pgpk -a KEYS
```

or

```
pgp -ka KEYS
```

To verify the binaries/sources you can download the relevant asc files for it from main distribution directory and follow the below guide.

```
gpg --verify apache-shardingsphere-********.asc apache-shardingsphere-elasticjob-
*********
```

or

```
pgpv apache-shardingsphere-elasticjob-********.asc
```

or

```
pgp apache-shardingsphere-elasticjob-********.asc
```

9 Powered By

9.1 Register

Welcome to register by company + homepage + use case(optional), your support is important to us.

Please register here with `company` + `homepage` + `use case(optional)`.

9.2 Who are using ElasticJob?

Total: 83 companies.

9.2.1 E-commerce

DangDang

Three Squirrels

BESSKY

HAI ZOL

Xiu

homedo

AVIC B2B ONLINE TRADING OLATRORM

GShopper

ChunBo

HuiNong

DaZong

YangSC

DG-Mail

Nex Poster

JD

9.2.2 Financial Industry

Best Pay

WX XISHANG BANK

ppdai

YinSheng E-Pay

ZhongAn Tech

JinCaiHuLian

Lianlian Pay

SR online

IcInfo

LaoCaiBao

NiuCard

JieDaiBao

JinHui365

91 Tech Group

9.2.3 Digitalization and Cloud Services

YunJia cloud

Joyowo

Tree Bear

南方电网深圳数研院

9.2.4 Transportation

JUNEYAO AIR

CaoCao

Tuhu

ShouQi

iTrip

MaiHaoche

TTPai

DiDi

9.2.5 Logistics

YR Express

HaoYunHu

DeKun

9.2.6 Real Estate

ZIroom

UCommune

LianJia

9.2.7 E-education

IBeiLiao

IQiHang

Will Class

Think Town

GSX

Qidian

9.2.8 E-entertainment

MiguFun

motie

squirrel

9.2.9 News

FangJia

IFeng

Taoguba

FanHaoYue

SOHU

9.2.10 Communication

MeiZu

OnePlus

9.2.11 Internet of Things

Lenovo

Neoway

Gizwits

YY Cloud

G7

ShenzhenGuangliansaixun Co.， LTD.

Guangzhou shang mai network technology co. LTD

9.2.12 Software Development Services

ultrapower

DuiBa Group

Cig

Yeahmobi

LeiMing

ZhongChuang Technology

DeepDraw

WeiLaiXinFeng

Guangzhou Zhongruan

PubLink

9.2.13 Health Care

H&H Global

Glory

YIBAO

SYTown

9.2.14 Retail

YH

9.2.15 AI

DeepBlue

10 FAQ

10.1 Why do some compiling errors appear?

Answer:

ElasticJob uses lombok to enable minimal coding. For more details about using and installment, please refer to the official website of lombok.

10.2 Does ElasticJob support dynamically adding jobs?

Answer:

For the concept of dynamically adding job, everyone has a different understanding.

ElasticJob-Lite is provided in jar package, which is started by developers or operation. When the job is started, it will automatically register job information to the registry center, and the registry center will perform distributed coordination, so there is no need to manually add job information in the registry center. However, registry center has no affiliation with the job server, can't control the distribution of single-point jobs to other job machines, and also can't start the job of remote server. ElasticJob-Lite doesn't support ssh secret management and other functions.

ElasticJob-Cloud is a mesos framework, and mesos is responsible for job starting and distribution. But you need to package the job and upload it, and call the REST API provided by ElasticJob-Cloud to write job information into the registry center. Packaging and uploading job are the deployment system's functions, ElasticJob-Cloud does not support it.

In summary, ElasticJob has supported basic dynamically adding jobs, but it can't be fully automated.

10.3 Why is the job configuration modified in the code or Spring XML file, but the registry center is not updated?

Answer:

ElasticJob-Lite adopts a decentralized design. If the configuration of each client is inconsistent and is not controlled, the configuration of the client which is last started will be the final configuration of the registry center.

ElasticJob-Lite proposes the concept of overwrite, which can be configured through JobConfiguration or Spring namespace. overwrite=true indicates that the client's configuration is allowed to override the registry center, and on the contrary is not allowed. If there is no configuration of related jobs in the registry center, regardless of whether the property of overwrite is configured, the client's configuration will be still written into the registry center.

10.4 What happens if the job can't communicate with the registry center?

Answer:

In order to ensure the consistency of the job in the distributed system, once the job can't communicate with the registry center, the job will stop immediately, but the job's process will not exit. The purpose of this is to prevent the assignment of the shards executed by the node that has lost contact with the registry center to another node when the job is re-sharded, causing the same shard to be executed on both nodes at the same time. When the node resumes contact with the registry center, it will re-participate in the sharding and resume execution of the newly shard.

10.5 What are the usage restrictions of ElasticJob-Lite?

Answer:

- After the job start successfully, modifying the job name is regarded as a new job, and the original job is discarded.
- It will be triggered re-sharding if the server changes, or if the sharding item is modified; re-sharding will cause the running streaming job to stop after the job is executed, and this job will return to normal after the re-sharding is finished.
- Enable monitorExecution to realize the function of distributed job idempotence (that is, the same shard will not be run on different job servers), but monitorExecution has a greater impact on the performance of jobs executed in a short period of time (such as second-level triggers). It is recommended to turn it off and realize idempotence by yourself.

10.6 What should you do if you suspect that ElasticJob-Lite has a problem in a distributed environment, but it cannot be reproduced and cannot be debugged in the online environment?

Answer:

Distributed problems are very difficult to debug and reproduce. For this reason, ElasticJob-Lite provides the dump command.

If you suspect a problem in some scenarios, you can refer to the dump document to submit the job runtime information to the community. ElasticJob has filtered sensitive information such as IP, and the dump file can be safely transmitted on the Internet.

10.7 What are the usage restrictions of ElasticJob-Cloud?

Answer:

- After the job start successfully, modifying the job name is regarded as a new job, and the original job is discarded.

10.8 When add a task in the ElasticJob-Cloud, why does it remain in the ready state, but doesn't start?

Answer:

The task will start when mesos has a separate agent that can provide the required resources, otherwise it will wait until there are enough resources.

10.9 Why can't the Console page display normally?

Answer:

Make sure that the Web Console's version is consistent with ElasticJob, otherwise it will become unavailable.

10.10 Why is the job state shard to be adjusted in the Console?

Answer:

Shard to be adjusted indicates the state when the job has started but has not yet obtained the shard.

10.11 Why is there a task scheduling delay in the first startup?

Answer:

ElasticJob will obtain the local IP when performing task scheduling, and it may be slow to obtain the IP for the first time. Try to set `-Djava.net.preferIPv4Stack=true`.

10.12 In Windows env, run ShardingSphere-ElasticJob-UI, could not find or load main class org.apache.shardingsphere.elasticjob.lite.ui.Bootstrap Why?

Answer:

Some decompression tools may truncate the file name when decompressing the ShardingSphere-ElasticJob-UI binary package, resulting in some classes not being found

Open cmd.exe and execute the following command:

```
tar zxvf apache-shardingsphere-elasticjob-${RELEASE.VERSION}-lite-ui-bin.tar.gz
```

10.13 Unable to startup Cloud Scheduler. Continuously output "Elastic job: IP:PORT has leadership"

Answer:

Cloud Scheduler required Mesos native library. Specify Mesos native library path by property `-Djava.library.path`.

For instance, Mesos native libraries are under `/usr/local/lib`, so the property `-Djava.library.path=/usr/local/lib` need to be set to start the Cloud Scheduler.

About Apache Mesos, please refer to Apache Mesos.

10.14 Unable to obtain a suitable IP in the case of multiple network interfaces

Answer:

You may specify interface by system property `elasticjob.preferred.network.interface` or specify IP by system property `elasticjob.preferred.network.ip`.

For example

1. specify the interface eno1: `-Delasticjob.preferred.network.interface=eno1`.
2. specify network addresses, 192.168.0.100: `-Delasticjob.preferred.network.ip=192.168.0.100`.
3. specify network addresses for regular expressions, 192.168.*: `-Delasticjob.preferred.network.ip=192.168.*`.

11

Blog

TODO