# 1.作业分片策略

## 1框架提供的分片策略

### 1.1 AverageAllocationJobShardingStrategy

#### 1.1.1全路径

com.dangdang.ddframe.job.plugin.sharding.strategy.AverageAllocationJobShardingStrategy

#### 1.1.2策略说明

基于平均分配算法的分片策略，也是默认的分片策略。

如果分片不能整除，则不能整除的多余分片将依次追加到序号小的服务器。

如：

如果有3台服务器，分成9片，则每台服务器分到的分片是：1=[0,1,2], 2=[3,4,5], 3=[6,7,8]

如果有3台服务器，分成8片，则每台服务器分到的分片是：1=[0,1,6], 2=[2,3,7], 3=[4,5]

如果有3台服务器，分成10片，则每台服务器分到的分片是：1=[0,1,2,9], 2=[3,4,5], 3=[6,7,8]

### 1.2 OdevitySortByNameJobShardingStrategy

#### 1.2.1全路径

com.dangdang.ddframe.job.plugin.sharding.strategy.OdevitySortByNameJobShardingStrategy

#### 1.2.2策略说明

根据作业名的哈希值奇偶数决定IP升降序算法的分片策略。

作业名的哈希值为奇数则IP升序。

作业名的哈希值为偶数则IP降序。

用于不同的作业平均分配负载至不同的服务器。

AverageAllocationJobShardingStrategy的缺点是，一旦分片数小于作业服务器数，作业将永远分配至IP地址靠前的服务器，导致IP地址靠后的服务器空闲。而OdevitySortByNameJobShardingStrategy则可以根据作业名称重新分配服务器负载。

如：

如果有3台服务器，分成2片，作业名称的哈希值为奇数，则每台服务器分到的分片是：1=[0], 2=[1], 3=[]

如果有3台服务器，分成2片，作业名称的哈希值为偶数，则每台服务器分到的分片是：3=[0], 2=[1], 1=[]

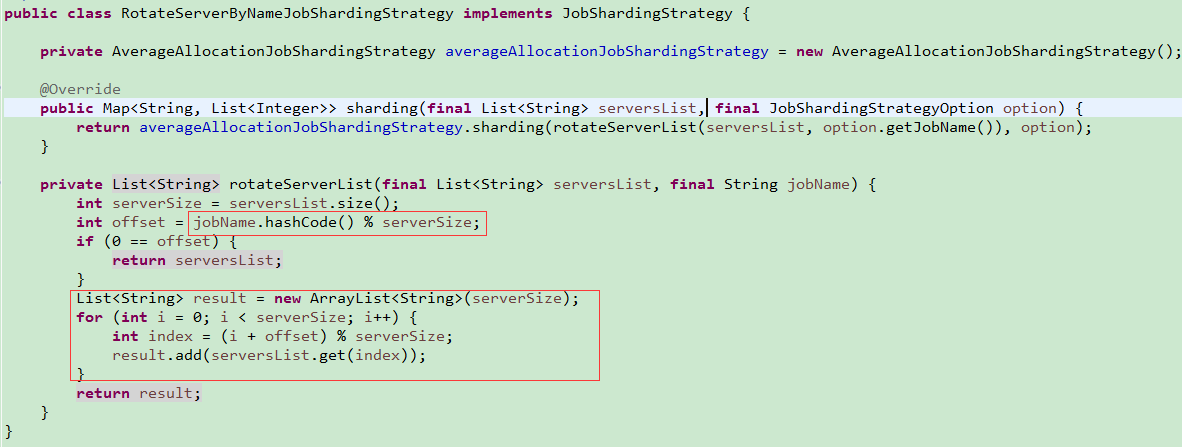
### 1.3 RotateServerByNameJobShardingStrategy

#### 1.3.1全路径

com.dangdang.ddframe.job.plugin.sharding.strategy.RotateServerByNameJobShardingStrategy

#### 1.3.2策略说明

根据作业名的哈希值对服务器列表进行轮转的分片策略。



## 2自定义分片策略

实现JobShardingStrategy接口并实现sharding方法，接口方法参数为作业服务器IP列表和分片策略选项，分片策略选项包括作业名称，分片总数以及分片序列号和个性化参数对照表，可以根据需求定制化自己的分片策略。

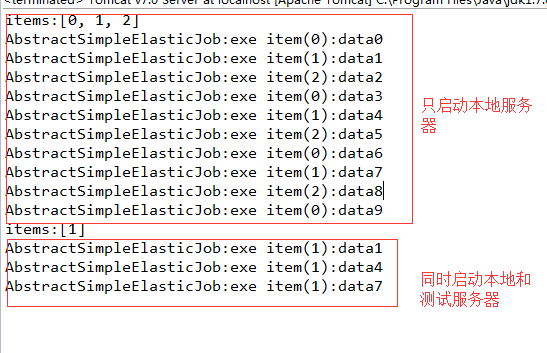
## 3配置分片策略

与配置通常的作业属性相同，在spring命名空间或者JobConfiguration中配置jobShardingStrategyClass属性，属性值是作业分片策略类的全路径。

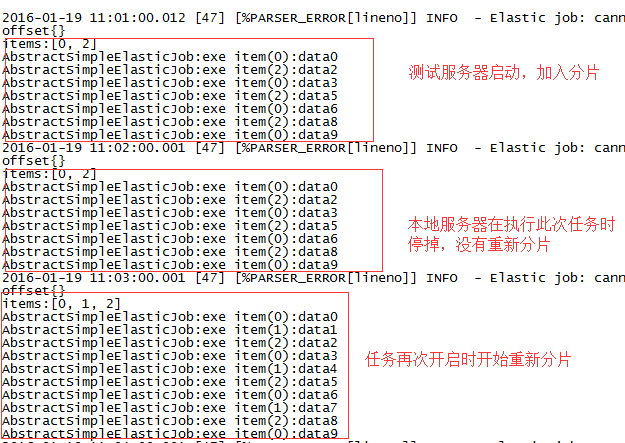
# 2.弹性扩容测试

失效转移默认为false

本地服务器日志：



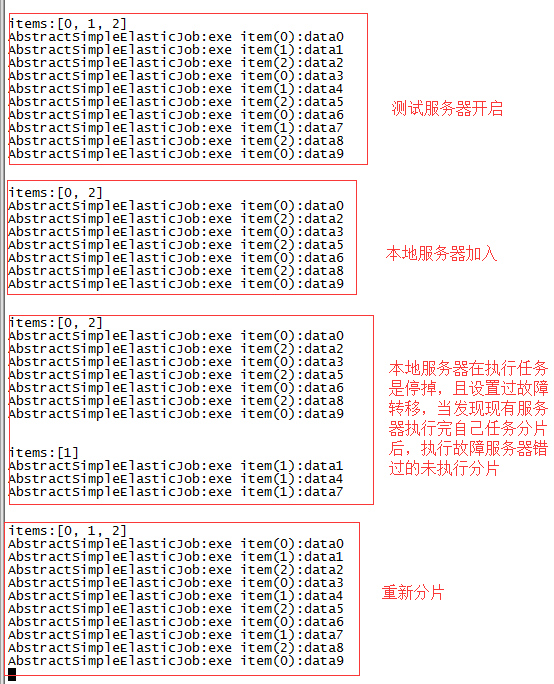
测试服务器日志



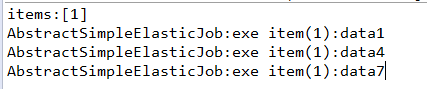
# 3.失效转移测试

失效转移：failover=*"true"*

测试服务器先开启：



本地服务器：



# 4.作业执行模式

目前提供2种作业类型，分别是Simple和DataFlow。DataFlow类型用于处理数据流，它又提供2种作业类型，分别是ThroughputDataFlow和SequenceDataFlow。需要继承相应的抽象类。

方法参数shardingContext包含作业配置，分片和运行时信息。可通过getShardingTotalCount(),getShardingItems()等方法分别获取分片总数，运行在本作业服务器的分片序列号集合等。

## 4.1Simple类型作业

Simple类型作业意为简单实现，未经任何封装的类型。需要继承AbstractSimpleElasticJob，该类只提供了一个方法用于覆盖，此方法将被定时执行。用于执行普通的定时任务，与Quartz原生接口相似，只是增加了弹性扩缩容和分片等功能。



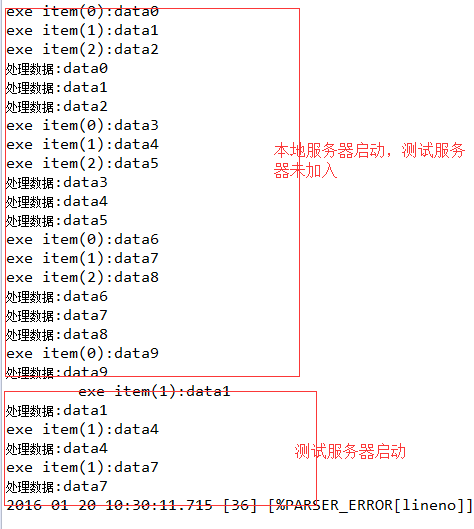
## 4.2ThroughputDataFlow类型作业

ThroughputDataFlow类型作业意为高吞吐的数据流作业。需要继承AbstractThroughputDataFlowElasticJob并可以指定返回值泛型，该类提供3个方法可覆盖，分别用于抓取数据，处理数据和指定是否流式处理数据。可以获取数据处理成功失败次数等辅助监控信息。如果流式处理数据，fetchData方法的返回值只有为null或长度为空时，作业才会停止执行，否则作业会一直运行下去；非流式处理数据则只会在每次作业执行过程中执行一次fetchData方法和processData方法，即完成本次作业。流式数据处理参照TbSchedule设计，适用于不间歇的数据处理。

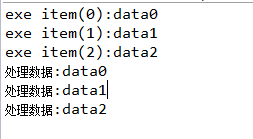
作业执行时会将fetchData的数据传递给processData处理，其中processData得到的数据是通过多线程（线程池大小可配）拆分的。如果采用流式作业处理方式，建议processData处理数据后更新其状态，避免fetchData再次抓取到，从而使得作业永远不会停止。processData的返回值用于表示数据是否处理成功，抛出异常或者返回false将会在统计信息中归入失败次数，返回true则归入成功次数。



1.流式处理（多次抓取数据，直到抓取返回值为null或长度为空时）



2.非流式处理（只抓取一次数据，抓取多少数据，处理多少数据）



## 4.3SequenceDataFlow类型作业

SequenceDataFlow类型作业和ThroughputDataFlow作业类型极为相似，所不同的是ThroughputDataFlow作业类型可以将获取到的数据多线程处理，但不会保证多线程处理数据的顺序。如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，配置为两个线程处理，则第1个线程处理前50条数据，第2个线程处理后50条数据，无视分片项；SequenceDataFlow类型作业则根据当前服务器所分配的分片项数量进行多线程处理，每个分片项使用同一线程处理，防止了同一分片的数据被多线程处理，从而导致的顺序问题。如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，则系统自动分配两个线程处理，第1个线程处理第1个分片的40条数据，第2个线程处理第2个分片的60条数据。由于ThroughputDataFlow作业可以使用多于分片项的任意线程数处理，所以性能调优的可能会优于SequenceDataFlow作业。

public class MyElasticJob extends AbstractSequenceDataFlowElasticJob<Foo> {

@Override

public List<Foo> fetchData(JobExecutionSingleShardingContext context) {

int offset = context.getOffset();

List<Foo> result = // get data from database by sharding items and by offset

return result;

}

@Override

public boolean processData(JobExecutionSingleShardingContext context, Foo data) {

// process data

// ...

// store offset

updateOffset(context.getShardingItem(), "your offset, maybe id");

return true;

}

@Override

public boolean isStreamingProcess() {

return true;

}

}