

# 分布式调度

---

## 1 什么是分布式调度

---

分布式调度有两层含义：

1. 同一个任务在集群中部署多份，只应该有一个定时任务在执行
2. 把一个大任务拆分成多个小的作业任务，即分片，小任务同时执行

## 2 定时任务与消息队列的区别

---

共同点：

- 异步处理
- 应用解耦
- 流量削峰

不同点：

定时任务是时间驱动，即每隔一定时间都要去干这件事；

消息队列是事件驱动，即活儿来了，我再开始干；

## 3 分布式调度框架Elastic-Job

---

### 3.1 简介

Elastic-Job是当当网开源的一个分布式调度解决方案，基于Quartz二次开发的，由两个相互独立的子项目Elastic-Job-Lite和Elastic-Job-Cloud组成。

Elastic-Job-Lite，它定位为轻量级无中心化解决方案，使用Jar包的形式提供分布式任务的协调服务。

Elastic-Job-Cloud子项目需要结合Mesos以及Docker在云环境下使用。

### 3.2 功能

- 分布式调度协调  
在分布式环境中，任务能够按指定的调度策略执行，并且**能够避免同一任务多实例重复执行**。
- 丰富的调度策略  
基于成熟的定时任务作业框架Quartz cron表达式执行定时任务。
- 弹性扩容缩容  
当集群中增加某一个实例，它应当也能够被选举并执行任务；当集群减少一个实例时，它所执行的任务能被转移到别的实例来执行。
- 失效转移  
某实例在任务执行失败后，会被转移到其他实例执行。
- 错过执行作业重触发  
若因某种原因导致作业错过执行，自动记录错过执行的作业，并在上次作业完成后自动触发。
- 支持任务分片

当任务被分片后，保证同一分片在分布式环境中仅一个执行实例。

### 3.3 使用

- 导入Elastic-Job依赖

```
<dependency>
  <groupId>com.dangdang</groupId>
  <artifactId>elastic-job-lite-core</artifactId>
  <version>2.1.5</version>
</dependency>
```

- 配置并启动zookeeper，把zookeeper/conf/目录下的zoo\_sample.cfg复制一份重命名为zoo.cfg，在zookeeper/bin目录下运行zkServer.cmd

此电脑 > 本地磁盘 (D:) > apache-zookeeper-3.6.0-bin > conf

名称	修改日期
configuration.xml	2020/2/25 22:36
log4j.properties	2020/2/25 22:36
zoo.cfg	2020/2/25 22:36
zoo_sample.cfg	2020/2/25 22:36

- 编写调度任务类：

这里模拟一个场景，给小朋友分发糖果，小朋友有性别之分，用于之后测试任务分片

创建CandyJob类继承SimpleJob接口并实现execute()方法

分发逻辑为：每次从所有小朋友中选出没有糖果且对应分片任务性别的小朋友，作为分发集合，然后给每个小朋友发糖果。发糖果逻辑就是修改小朋友实体类Kid的haveCandy字段为true

```
public class CandyJob implements SimpleJob {
    public static List<Kid> kids = new ArrayList<Kid>();
    private final int KIDS_COUNT = 1;

    public void execute(ShardingContext shardingContext) {
        int shardingItem = shardingContext.getShardingItem();
        System.out.println("当前分片为=====》分片: " + shardingItem);
        //获取分片参数
        String shardingParameter = shardingContext.getShardingParameter();

        //获取没有糖果的孩子列表，不分片
        //List<Kid> out = whoDoesntGet(KIDS_COUNT);

        //获取没有糖果的孩子列表，分片
        List<Kid> out = whoDoesntGet(KIDS_COUNT, shardingParameter);
        //分发糖果
        handOut(out);
    }

    /**
     * 按照性别来分发糖果，测试分片
     */
}
```

```

    * @param count
    * @param gender
    * @return
    */
    public List<Kid> whoDoesntGet(int count, String gender) {
        List<Kid> out = new ArrayList<Kid>();
        int n = 0;
        for (Kid kid : kids) {
            if (n >= count) {
                break;
            }
            if (!kid.isHaveCandy() && kid.getGender().equals(gender)) {
                out.add(kid);
                n++;
            }
        }
        return out;
    }

    /**
     * 分发糖果
     * @param kids
     */
    public void handOut(List<Kid> kids) {
        for (Kid kid : kids) {
            kid.setHaveCandy(true);
            System.out.println("给小朋友: " + kid.toString() + "分发了糖果");
        }
    }

    /**
     * 获取未得到糖果的kids列表
     * @param count 一次性获取几个kids
     * @return 未得到糖果的kids列表
     */
    public List<Kid> whoDoesntGet(int count) {
        List<Kid> out = new ArrayList<Kid>();
        int n = 0;
        for (Kid kid : kids) {
            if (n >= count) {
                break;
            }
            if (!kid.isHaveCandy()) {
                out.add(kid);
                n++;
            }
        }
        return out;
    }
}

```

- 实体类Kid如下

```

public class Kid {
    private String name;
    private String gender;
    private boolean haveCandy=false;
}

```

```

    public Kid(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isHaveCandy() {
        return haveCandy;
    }

    public void setHaveCandy(boolean haveCandy) {
        this.haveCandy = haveCandy;
    }

    @Override
    public String toString() {
        return "Kid{" +
            "name='" + name + '\'' +
            ", gender='" + gender + '\'' +
            ", haveCandy=" + haveCandy +
            '}';
    }
}

```

- 接着创建任务启动类

在启动类main函数中，要进行测试数据创建、初始化zookeeper、任务配置以及创建调度对象等工作。

```

public class JobMain {

    public static void main(String[] args) {
        //制造一些测试数据
        String[] gender = {"male", "female"};
        for (int i = 0; i < 100; i++) {
            int sex = (int) (Math.random() * 2);
            CandyJob.kids.add(new Kid("kid:" + i, gender[sex]));
        }
        System.out.println("初始化测试数据完成");

        // 配置注册中心zookeeper， zookeeper协调调度，不能让任务重复执行，
    }
}

```

```

// 通过命名空间分类管理任务，对应到zookeeper的目录
ZookeeperConfiguration zookeeperConfiguration =
    new ZookeeperConfiguration("localhost:2181", "candy-job");
CoordinatorRegistryCenter coordinatorRegistryCenter =
    new ZookeeperRegistryCenter(zookeeperConfiguration);
coordinatorRegistryCenter.init();

//任务配置
JobCoreConfiguration jobCoreConfiguration =
    JobCoreConfiguration.newBuilder(
        "archive-job",
        "*/3 * * * * ?",//每隔3秒
        2).//设置任务分片数量
        shardingItemParameters("0=male,1=female").//分片参数
        build();
SimpleJobConfiguration simpleJobConfiguration =
    new SimpleJobConfiguration(jobCoreConfiguration,
        CandyJob.class.getName());

//创建调度对象
new JobScheduler(coordinatorRegistryCenter,

    LiteJobConfiguration.newBuilder(simpleJobConfiguration).overwrite(true).build()).init();
    }
}

```

映射

- 测试分布式调度

启动两个jobMain进程，观察控制台：

```
JobMain × JobMain ×
Debugger Console
当前分片为=====》分片：0
给小朋友：Kid{name='kid:2', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:3', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:4', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:5', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:7', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:10', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:11', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:12', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:14', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:15', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:16', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:17', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片：0
给小朋友：Kid{name='kid:18', gender='male', haveCandy=true}分发了糖果
```

```
JobMain x JobMain x
Debugger Console
当前分片为=====》分片:1
给小朋友:Kid{name='kid:8', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:9', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:10', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:17', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:18', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:21', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:23', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:25', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:27', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:30', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:31', gender='female', haveCandy=true}分发了糖果
```

可以看到，两个任务分片分别执行各自性别的小朋友的糖果分发，证明很好的实现了任务分片。

接着我们停掉其中一个进程，观察控制台：

```
当前分片为=====》分片:1
给小朋友:Kid{name='kid:14', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:0
给小朋友:Kid{name='kid:4', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:15', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:0
给小朋友:Kid{name='kid:5', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:16', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:0
给小朋友:Kid{name='kid:6', gender='male', haveCandy=true}分发了糖果
当前分片为=====》分片:1
给小朋友:Kid{name='kid:20', gender='female', haveCandy=true}分发了糖果
当前分片为=====》分片:0
给小朋友:Kid{name='kid:9', gender='male', haveCandy=true}分发了糖果
```

可以看到，所有分片都由存活的进程执行，证明了Elastic-Job的高可用。

## 4 Elastic-Job-Lite轻量级去中心化的特点

### 4.1 去中心化

Elastic-Job中的所有节点都是平等的，不存在一个中心调度节点，这里的概念和zookeeper有区别，zookeeper作用是注册中心，把一些服务注册上去，并不会指派给谁。而Elastic-Job节点具备服务自发现，当发现注册中心有服务注册时，任务节点便可以去执行。

## 4.2 轻量级

Elastic-Job不需要独立部署，不属于服务中间件，使用时只需引入相关jar包（依赖），配合zookeeper使用即可。