当当 Elastic-job 开源项目的十项特性(1)

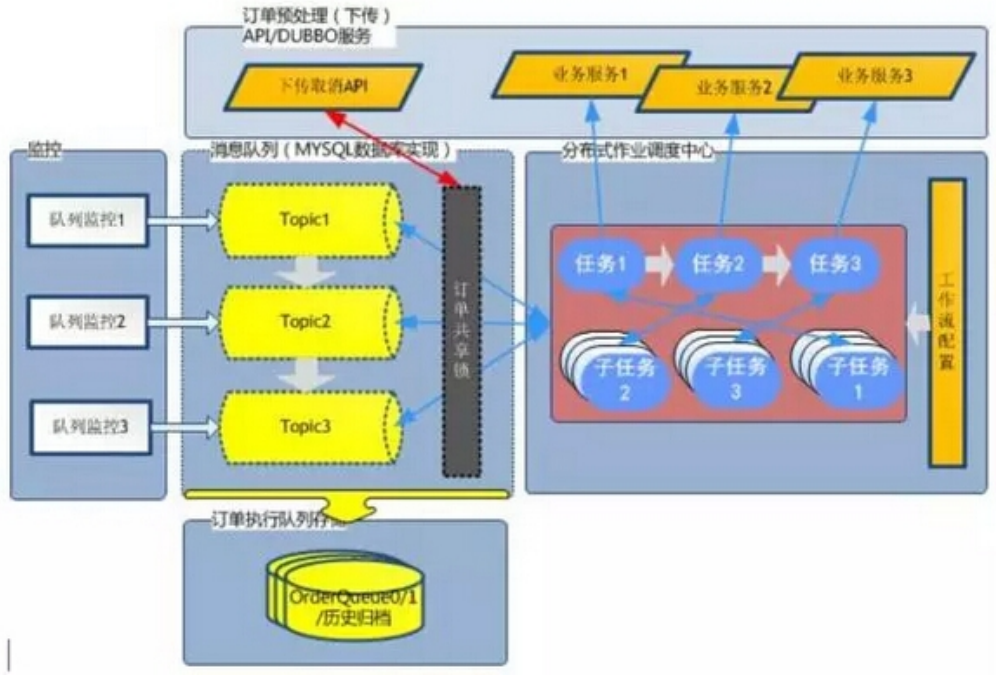
2015-11-05 10:51 张亮 高可用架构公众号 字号： T | T

Elastic-job原本是当当Java应用框架ddframe的一部分，本名dd-job。ddframe包括编码规范，开发框架，技术规范，监控以及分布式组件。ddframe规划分为4个演进阶段，目前处于第2阶段。3、4阶段涉及的技术组件不代表当当没有使用，只是ddframe还未统一规划。

张亮：当当网架构师、当当技术委员会成员、消息中间件组负责人。对架构设计、分布式、优雅代码等领域兴趣浓厚。目前主导当当应用框架ddframe研发，并负责推广及撰写技术白皮书。

一、为什么需要作业(定时任务)?

作业即定时任务。一般来说，系统可使用消息传递代替部分使用作业的场景。两者确有相似之处。可互相替换的场景，如队列表。将待处理的数据放入队列表，然后使用频率极短的定时任务拉取队列表的数据并处理。这种情况使用消息中间件的推送模式可更好的处理实时性数据。而且基于数据库的消息存储吞吐量远远小于基于文件的顺序追加消息存储。



但在某些场景下则不能互换：

1. 时间驱动 OR 事件驱动：内部系统一般可以通过事件来驱动，但涉及到外部系统，则只能使用时间驱动。如：抓取外部系统价格。每小时抓取，由于是外部系统，不能像内部系统一样发送事件触发事件。

2. 批量处理 OR 逐条处理：批量处理堆积的数据更加高效，在不需要实时性的情况下比消息中间件更有优势。而且有的业务逻辑只能批量处理，如：电商公司与快递公司结算，一个月结算一次，并且根据送货的数量有提成。比如，当月送货超过1000则额外给快递公司多1%优惠。

3. 非实时性 OR 实时性：虽然消息中间件可以做到实时处理数据，但有的情况并不需要如的实时。如：VIP用户降级，如果超过1年无购买行为，则自动降级。这类需求没有强烈的时间要求，不需要按照时间精确的降级VIP用户。

4. 系统内部 OR 系统解耦。作业一般封装在系统内部，而消息中间件可用于系统间解耦。

二、当当之前在使用什么作业系统?

当当之前使用的作业系统比较散乱，各自为战，大致分为以下4种：

1. Quartz：Java事实上的定时任务标准。但Quartz关注点在于定时任务而非数据，并无一套根据数据处理而定制化的流程。虽然Quartz可以基于数据库实现

作业的高可用，但缺少分布式并行执行作业的功能。

2. TBSchedule：阿里早期开源的分布式任务调度系统。代码略陈旧，使用timer而非线程池执行任务调度。众所周知，timer在处理异常状况时是有缺陷的。而且TBSchedule作业类型较为单一，只能是获取/处理数据一种模式。还有就是文档缺失比较严重。

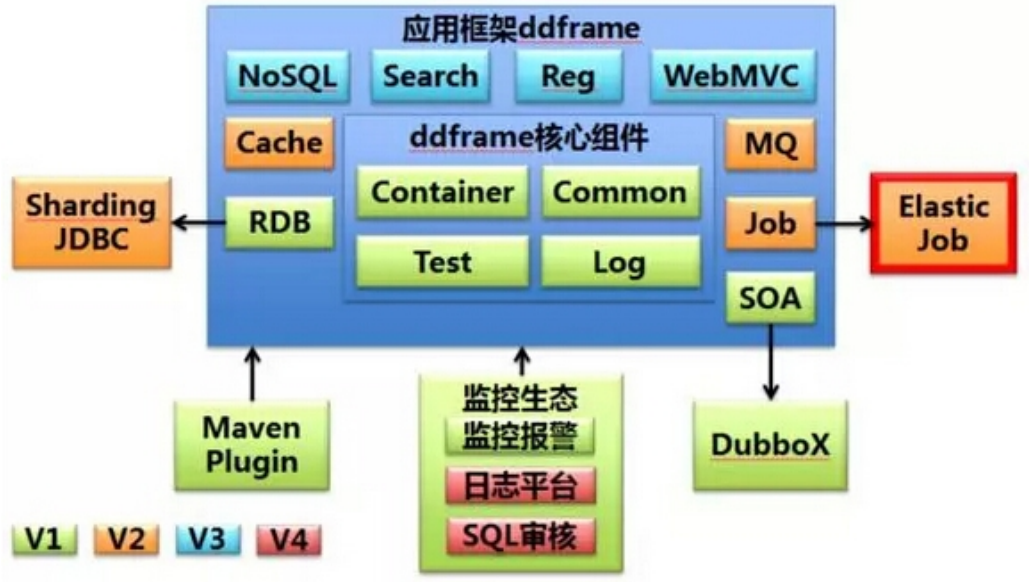
3. Crontab：Linux系统级的定时任务执行器。缺乏分布式和集中管理功能。

4. Perl：遗留系统使用，目前已不符合公司的Java化战略。

三、elastic-job的来历

elastic-job原本是当当Java应用框架ddframe的一部分，本名dd-job。

ddframe包括编码规范，开发框架，技术规范，监控以及分布式组件。ddframe规划分为4个演进阶段，目前处于第2阶段。3、4阶段涉及的技术组件不代表当当没有使用，只是ddframe还未统一规划。



ddframe由各种模块组成，均已dd-开头，如dd-container，dd-soa，dd-rdb，dd-job等。当当希望将ddframe的各个模块与公司环境解耦并开源以反馈社区。之前开源的Dubbo扩展版本DubboX即是dd-soa的核心模块。而本次介绍的elastic-job则是dd-job的开源部分，其中监控(但开源了监控方法)和ddframe核心接入等部分并未开源。

四、elastic-job包含的功能

- 分布式：最重要的功能，如果任务不能在分布式的环境下执行，那么直接

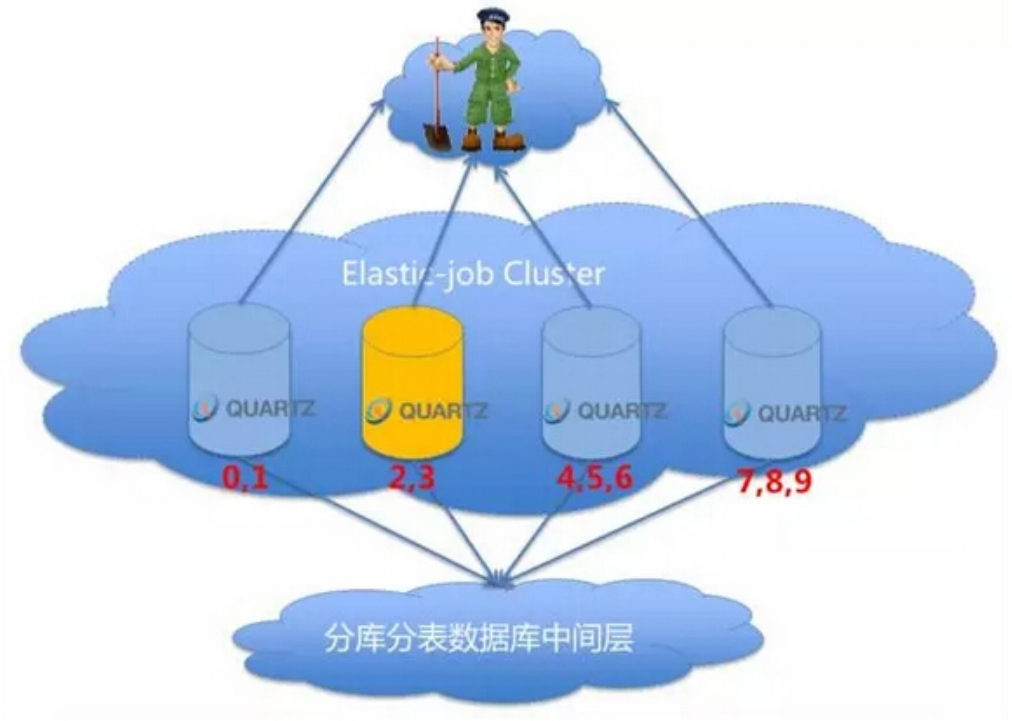
使用Quartz就可以了。

- 任务分片：是elastic-job中最重要也是最难理解的概念。任务的分布式执行，需要将一个任务拆分为n个独立的任务项，然后由分布式的服务器分别执行某一个或几个分片项。
- 弹性扩容缩容：将任务拆分为n个任务项后，各个服务器分别执行各自分配到的任务项。一旦有新的服务器加入集群，或现有服务器下线，elastic-job将在保留本次任务执行不变的情况下，下次任务开始前触发任务重分片。举例说明：有3台服务器，分为10个片。则分片项分配如下：{server1: [0,1,2], server2: [3,4,5], server3: [6,7,8,9]}。如果一台服务器崩溃，则分片项分配如下：{server1: [0,1,2,3,4], server2: [5,6,7,8,9]}。如果新增一台服务器，则分片项分配如下：{server1: [0,1], server2: [2,3] , server3: [4,5,6] , server4: [7,8,9]}。
- 稳定性：在服务器无波动的情况下，并不会重新分片;即使服务器有波动，下次分片的结果也会根据服务器IP和作业名称哈希值算出稳定的分片顺序，尽量不做大的变动。
- 高性能：elastic-job会将作业运行状态的必要信息更新到注册中心，但为了考虑性能问题，可以牺牲一些功能，而换取性能的提升。
- 幂等性：elastic-job可牺牲部分性能用以保证同一分片项不会同时在两个服务器上运行。
- 失效转移：弹性扩容缩容在下次作业运行前重分片，但本次作业执行的过程中，下线的服务器所分配的作业将不会重新被分配。失效转移功能可以在本次作业运行中用空闲服务器抓取孤儿作业分片执行。同样失效转移功能也会牺牲部分性能。
- 状态监控：监控作业的运行状态，可以监控数据处理功能和失败次数，作业运行时间等。是幂等性，失效转移必须的功能。
- 多作业模式：作业可分为简单和数据流处理两种模式，数据流又分为高吞吐处理模式和顺序性处理模式，其中高吞吐处理模式可以开启足够多的线程快速的处理数据，而顺序性处理模式将每个分片项分配到一个独立线程，用于保证同一分片的顺序性，这点类似于kafka的分区顺序性。
- 其他一些功能，如错过任务重执行，单机并行处理，容错处理，Spring命名

空间支持，运维平台等。

五、elastic-job的部署和使用

将使用elastic-job框架的jar/war连接同一个基于Zookeeper的注册中心即可。



作业框架执行数据并不限于数据库，且作业框架本身是不对数据进行关联的。作业可以用于处理数据，文件，API等任何操作。

使用elastic-job所需要关注的仅仅是将业务处理逻辑和框架所分配的分片项匹配并处理，如：如果分片项是1，则获取id以1结尾的数据处理。所以如果是处理数据的话，最佳实践是将作业分片项规则和数据中间层规则对应。

通过上面的部署图可以看出来，作业分片只是个逻辑概念，分片和实际数据其实框架是不做任何匹配关系的。而根据分片项和实际业务如何关联，是成功使用elastic-job的关键所在。为了不让代码写起来很无聊，看起来像if(shardingItem == 1) {do xxx} else if (shardingItem == 2) {do xxx}，elastic-job提供了自定义参数，可将分片项序号和实际业务做映射。比如设置为1=北京，2=上海。那么代码中可以通过北京或是上海的枚举，从业务中的北京仓库或上海仓库取数据。elastic-job更多的还是关注作业调度和分布式分配，处理数据还是交由数据中间层更好些。

诚如刚才所说，最佳实践是将作业分片项规则和数据中间层规则对应，省去

作业分片时，再次适配数据中间层的分片逻辑。

六、对开源产品的开发理念

为了让感兴趣的人放心使用，我想分享一下我们对开源产品的开发理念。

用心写代码。代码是项目的唯一核心和产出，任何一行的代码都需要用心思考优雅性，可读性，合理性。优雅性看似简单的几个字，其实实现的难度非常大。每个人心中都有自己对代码的理解，而elastic-job也好，ddframe也好，都不是出自一人之手。对代码优雅性的权衡，是比较难把控的。后面几项，可以理解为对第一项的补充，或具体的实现思路。

代码整洁干净到极致。简单点说就是重度代码洁癖患者。只有代码漂亮整洁，其他开源爱好者才愿意阅读代码，进而找出项目中的bug和贡献高质量代码。

极简代码，高度复用，无重复代码和配置。Java生态圈的特点是高质量的开源产品极多。我们尽量考虑复用轮子，比如项目中大量用到lombok简化代码;但也不会无原则的使用开源产品，我们倾向于把开源产品分为积木类和大厦类。项目中一般只考虑使用积木类搭建属于我们自己的大厦，而不会直接用其他已成型的大厦。java系的公司有两种不同的声音，拥抱开源，或完全不使用开源。我们的看法是既然选择使用java，就应该遵循java的理念，去拥抱java这些年累积的成熟东西。java相比其他新兴语言，在语法上可能没什么优势，但在广度上还是少有其他生态圈可比拟。

单一需求可不考虑扩展性;两个类似需求时再提炼。为了不盲目追求所谓的极致，我们用这条规则，尽量提升交付的速度。

模块抽象划分合理。这点也很难用标准衡量。以elastic-job举例：elastic-job核心代码分为4块，core，spring，console和example;分别用于放置核心，spring支持，控制台和代码示例。在项目级别上做拆分。而core中将包分为api，exception，plugin和internal。用于放置对外发布的接口、异常，向最终用户提供的可扩展插件以及封装好的内部实现。内部实现的任何改动，都不会影响对外接口的变动，用户自定义的插件，也不会影响内部代码的稳定性。

如无特殊理由，测试需全覆盖。elastic-job核心模块的测试覆盖率是95%以上。

虽然单元测试覆盖率在分布式的复杂环境中并无太大说服力，但至少证明项目中很少出现低级逻辑错误。

对质量的定义。代码可读性 > 代码可测性 > 模块解耦设计 > 功能正确性 > 性能 > 功能可扩展性。只有代码可读，可测试，可100%掌控，项目才可持续发展。功能有缺陷可以修复，性能不够可以优化，而代码不清晰则项目会渐渐变为黑盒。所以对于框架类产品，我们认为质量 > 时间 > 成本。

文档清晰。

七、未来展望

监控体系有待提高，目前只能通过注册中心做简单的存活和数据积压监控。未来需要做的监控部分有：

1. 增加可监控维度，如作业运行时间等。

2. 基于JMX的内部状态监控。

3. 基于历史的全量数据监控，将所有监控数据通过flume等形式发到外部监控中心，提供实时分析功能。

增加任务工作流，如任务依赖，初始化任务，清理任务等。

失效转移功能的实时性提升。

更多作业类型支持，如文件，MQ等类型作业的支持。

更多分片策略支持。

项目的开源地址：https://github.com/dangdangdotcom/elastic-job 希望大家多关注，共同贡献代码。

Q&A

Q1：请问失效转移中如何判断失效?对任务本身实现有什么限制?

失效转移目前通过Zookeeper监听分片项临时节点判断。elastic-job会经过注册中心会话过期时间才能感知任务挂掉。失效转移有两种形式：1、任务挂掉，elastic-job会找空闲的作业服务器(可能是未分配任务的，也可能是完成执行本次任务执行的)执行。2、如果当时没有空闲服务器，则将在某服务器完成分配的任务时抓取未分配的分片项。

Q2：Zookeeper的作用是保存任务信息吗，如果Zookeeper挂了会影响任务执行吗?

Zookeeper目前的znode分四类，config，servers，execution，leader。config用于保存分布式作业的全局控制，如，分多少片，要不要执行misfire，cron表达式。servers用于注册作业服务器状态和分片信息。execution以分片的维度存储作业运行时状态。leader用于存储主节点。elastic-job作业执行是无中心化的，但主节点起到协调的作用，如：重分片、清理上次运行时信息等。

Q3：在任务处理上可以与spring batch集成吗?

spring batch之前关注过，但目前elastic-job还没有集成。elastic-job的spring支持是自定义了job的命名空间，更简化了基于spring的配置，并且可以使用spring注入的bean。spring batch也是很好的作业框架，包括spring-quartz也很不错，但分布式功能并不成熟。所以在这之上改动难度比较大，而且elastic-job更希望做一个不依赖于spring，而是能融入spring的绿色产品。

Q4：针对简单和数据流，能够说说具体分片是怎么处理的吗?

简单的作业就是未经过任何业务逻辑的封装，只是提供了一个execute方法，定时触发，但是增加了分布式分片功能。可以简单理解为quartz的分布式版本。quartz虽然可以支持基于数据库的分布式高可用，但不能分片。也就是说，两台服务器，只能一主一备，不能同时负载均衡的运行。数据流类型作业参照了阿里之前开源的TBSchedule，将数据处理分为fetchData和processData。先将数据从数据库，文件系统，或其他数据源取出来，然后processData集中处理，可以逐条处理，可以批量处理(这块未来将加上)。processData是多线程执行的，数据流类型作业可再细分为两种，一种是高吞吐，一种是顺序性。高吞吐可以开启任意多的线程并行执行数据处理，而顺序执行会根据每个分片项一个线程，保证分片项之中的数据有序，这点参照了kafka的实现。数据流类型作业有isStreaming这个参数，用于控制是否流式不停歇的处理数据，类似永动机，只要有数据，则一直处理。但这种作业不适合每次fetchData都对数据库造成压力很大的场景。

Q5：请问如何实现一个任务仅仅只在一个节点执行一次?

目前的幂等性，是在execution的znode中增加了对分片项状态的注册，如果状态是运行中，即使有别的服务器要运行这个分片项，elastic-job也会拒绝运行，而是等待这个状态变为非运行的状态。每个作业分片项启动时会更新状态。服务器没有波动的情况下，是不存在一个分片被分到两个服务器的情况。但一旦服务器

波动，在分片的瞬间有可能出现这种情况。关于分片，其实是比较复杂的实现。目前分片是发现服务器波动，或修改分片总数，将记录一个状态，而非直接分片。分片将在下次作业触发时执行，只有主节点可以分片，分片中从节点都将阻塞。无调度中心式分布式作业最大的一个问题是，无法保证主节点作业一定先于其他从节点触发。所以很有可能从节点先触发执行，而使用旧分片;然后主节点才重新分片，将造成这次作业分片可能不一致。这就需要execution节点来保证幂等性。下次执行时，只要无服务器波动，之前错误的分片自然会修正。

Q6：如果Zookeeper挂了，是否全部的任务都挂了不能运行包括已经运行过一次的，如果又恢复了，任务能正常运行吗，还是业务应用服务也要重新启动?

其实Zookeeper是不太容易挂的。毕竟Zookeeper是分布式高可用，一般不会是单台。目前elastic-job做到的容错是，连不上Zookeeper的作业服务器将立刻停止执行作业，防止主节点已重新分片，而脑裂的服务器还在执行。也就是说，Zookeeper挂掉，所有作业都将停止。而作业服务器一旦与Zookeeper恢复连接，作业也将恢复运行。所以Zookeeper挂掉不会影响数据，而Zookeeper恢复，作业会继续跑，不用重启。

Q7：可以具体到业务层面吗?比如有个任务，是一样发送100w的用户邮件，这时候应该怎么分片?针对分布式数据库的分页在咱们这里又是怎么处理的?

100W用户的邮件，个人认为可以按照用户id取模，比如分成100个分片，将整个userid % 100，然后每个分片发送userid结尾是取摸结果的邮件。详细来说：分片1发送以01结尾的userid的邮件，…，分片99发送以99结尾的userid的邮件。分布式数据库的分页，理论上来说，不是作业框架处理的范畴，应由数据中间层处理。顺便说下，ddframe的数据中间层部分，sharding-JDBC将于明年初开源。通过修改JDBC驱动实现分库分表。非MyCat或cobar这种中间件方式;也非基于hibernate或mybatis这种ORM方式。sharding-JDBC相对轻量级，也更加容易适配各种数据库和ORM

Q8：ddframe是由很多组件组成?支持多语言吗?

ddframe是很多组件的总称。分为核心模块，分布式组件模块和监控对接模块等。核心模块可以理解为spring-boot这种可快速启动，快速搭建项目的东西。

分布式组件包括SOA调用的Dubbox，基于分布式作业的elastic-job，还有刚才

提到的sharding-JDBC，以及近期暂无开源计划的缓存、MQ、NoSQL等模块。

监控模块估计以后也不会开源，和公司本身的业务场景绑定太紧，不是不想开源，是无法开源。主要分为日志中心，流量分析和系统关系调用图。监控部分目前也还在做，不是很强大。

多语言方面，SOA模块支持，Dubbox的REST扩展就是为了支持其他语言的调用。剩下的暂时不行。比如sharing-JDBC，主要是基于java的JDBC，如果多语言，中间层是个更好的方法。

ddframe的模块名字都是dd-*，dd-soa，dd-rdb，dd-job，dd-log之类。elastic-job，sharding-JDBC等，是为开源而从ddframe抽离并重新起的名字。

本文由张亮在高可用架构群所做的分享整理而来。转载请注明高可用架构公众号ArchNotes。