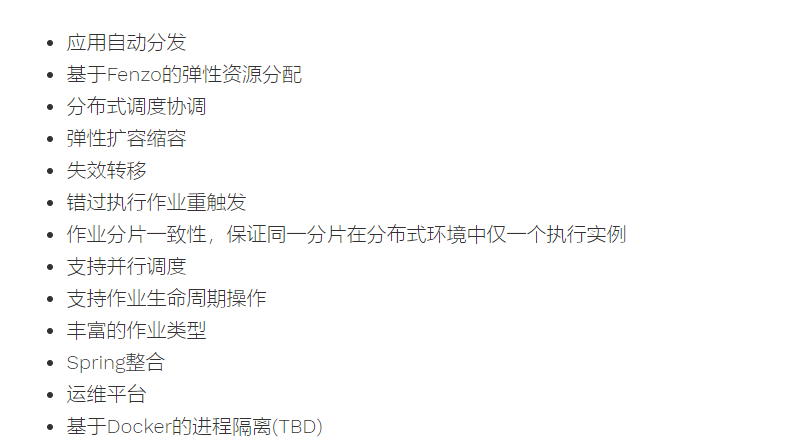
Elastic job是当当网架构师张亮,曹昊和江树建基于Zookepper、Quartz开发并开源的一个Java分布式定时任务，解决了Quartz不支持分布式的弊端。Elastic job主要的功能有支持弹性扩容，通过Zookepper集中管理和监控job，支持失效转移等，这些都是Quartz等其他定时任务无法比拟的。



### 1. 分片

任务的分布式执行，需要将一个任务拆分为多个独立的任务项，然后由分布式的服务器分别执行某一个或几个分片项。

数据分片的目的在于把一个任务分散到不同的机器上运行，既可以解决单机计算能力上限的问题，也能降低部分任务失败对整体系统的影响。elastic-job并不直接提供数据处理的功能，框架只会将分片项分配至各个运行中的作业服务器（其实是Job实例，部署在一台机器上的多个Job实例也能分片），开发者需要自行处理分片项与真实数据的对应关系。框架也预置了一些分片策略：平均分配算法策略，作业名哈希值奇偶数算法策略，轮转分片策略。同时也提供了自定义分片策略的接口。

例如：有一个遍历数据库某张表的作业，现有2台服务器。为了快速的执行作业，那么每台服务器应执行作业的50%。 为满足此需求，可将作业分成2片，每台服务器执行1片。作业遍历数据的逻辑应为：服务器A遍历ID以奇数结尾的数据；服务器B遍历ID以偶数结尾的数据。 如果分成10片，则作业遍历数据的逻辑应为：每片分到的分片项应为ID%10，而服务器A被分配到分片项0,1,2,3,4；服务器B被分配到分片项5,6,7,8,9，直接的结果就是服务器A遍历ID以0-4结尾的数据；服务器B遍历ID以5-9结尾的数据。

### 1. 分布式调度

Elastic-Job-Cloud采用Mesos Framework分片和协调作业调度。采用中心化调度实现难度小于Elastic-Job-Lite的无中心化调度，无需再考虑多线程并发的情况。

### 2. 作业高可用

Elastic-Job-Cloud由Mesos Framework负责作业高可用和分片。作业丢失会由Mesos Framework自动在另外的Agent上重新启动作业分片实例。

### 3. 弹性资源利用

Elastic-Job-Cloud分为2种作业运行模式：瞬时作业 和 常驻作业。

**瞬时作业会在每一次作业执行完毕后立刻释放资源，保证利用现有资源错峰执行**。资源分配和容器启动均占用一定时长，且作业执行时资源不一定充足，因此作业执行会有延迟。瞬时作业适用于间隔时间长，资源消耗多且对执行时间无严格要求的作业。

常驻作业无论在运行时还是等待运行时，均一直占用分配的资源，可节省过多容器启动和资源分配的开销，适用于间隔时间短，资源需求量稳定的作业。

##### 弹性扩容缩容

运行中的作业服务器崩溃，或新增加n台作业服务器，作业框架将在下次作业执行前重新分片，不影响当前作业执行。

##### 失效转移

Elastic-Job-Lite提供最安全的方式执行作业。将分片总数设置为1，并使用多于1台的服务器执行作业，作业将会以1主n从的方式执行。

一旦执行作业的服务器崩溃，等待执行的服务器将会在下次作业启动时替补执行。开启失效转移功能效果更好，可以保证在本次作业执行时崩溃，备机立即启动替补执行。

##### 错过执行作业重触发

自动记录错过执行的作业，并在上次作业完成后自动触发。

##### 作业分片一致性，保证同一分片在分布式环境中仅一个执行实例

##### 自诊断并修复分布式不稳定造成的问题

##### 支持并行调度

##### 支持作业生命周期操作

##### 丰富的作业类型

Elastic-Job提供Simple、Dataflow和Script 3种作业类型。

##### Spring整合以及命名空间提供

支持spring容器，自定义命名空间，支持占位符。

##### 运维平台

提供运维界面，可以管理作业和注册中心。

[**lastic-job详解（一）：数据分片**](http://www.cnblogs.com/haoxinyue/p/6919375.html)

数据分片的目的在于把一个任务分散到不同的机器上运行，既可以解决单机计算能力上限的问题，也能降低部分任务失败对整体系统的影响。elastic-job并不直接提供数据处理的功能，框架只会将分片项分配至各个运行中的作业服务器（其实是Job实例，部署在一台机器上的多个Job实例也能分片），开发者需要自行处理分片项与真实数据的对应关系。框架也预置了一些分片策略：平均分配算法策略，作业名哈希值奇偶数算法策略，轮转分片策略。同时也提供了自定义分片策略的接口。

<https://blog.csdn.net/thomas0yang/article/details/71549348>

<https://blog.csdn.net/lsshlsw/article/details/47086869>

<http://www.iocoder.cn/Elastic-Job/cloud-job-scheduler-and-executor-first/?sf&2017-10-31>

<https://blog.csdn.net/yinduntaian/article/details/72829464>

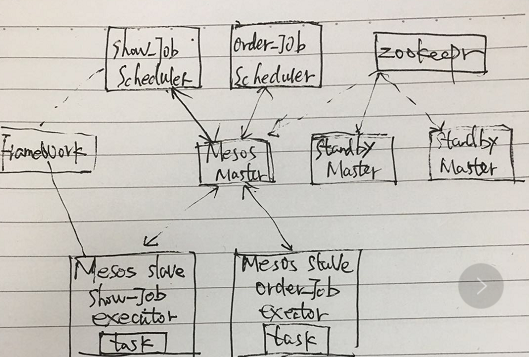
http://www.iocoder.cn/Elastic-Job/cloud-high-availability/?sf&2017-11-01

<https://blog.csdn.net/elim168/article/details/78732766>

<https://cloud.tencent.com/developer/news/208005>

<https://blog.csdn.net/boonya/article/details/51942040>

http://dongxicheng.org/apache-mesos/meso-architecture/



## 组件介绍

Mesos-master：协调全部的slave，并确定每个节点的可用资源，聚合计算跨节点的所有可用资源的报告，然后向注册到Master的Framework发出资源邀约。【资源管理、节点管理、接收资源申请、接收提交任务】

* Mesos-slave：向master汇报自己的空闲资源和任务的状态，负责管理本节点上的各个mesos-task，在framework成功向Master申请资源后,收到消息的slave会启动相应framework的exexutor【汇报资源、执行任务】
* Framework：实际干活的，如Hadoop，Spark等，通过MesosSchedulerDiver接入Mesos，可以理解为 mesos 上跑的应用，需要注册到 master 上。   
  一般包括两个主要组件：
* scheduler：调度器，注册到主节点，按照作业分解为任务，并申请资源并监控任务运行状况（类似 YARN 中的 ApplicationMaster）； executor：执行器，安装在 slave 节点上，负责执行本 framework的具体task。
* framework 分两种：一种是对资源需求可以 scale up 或者 down 的（Hadoop、Spark）；一种是对资源需求大小是固定的（MPI）。

## 双层调度

Master 根据一定策略（公平共享策略，或优先级策略【第一层】）决定为每个framework 提供多少资源， framework 的 scheduler 来选择其中提供的资源（框架内调度算法【第二层】）调度执行任务。当 framework 同意了提供的资源，它通过 master 将 task发送到提供资源的slaves 上运行。

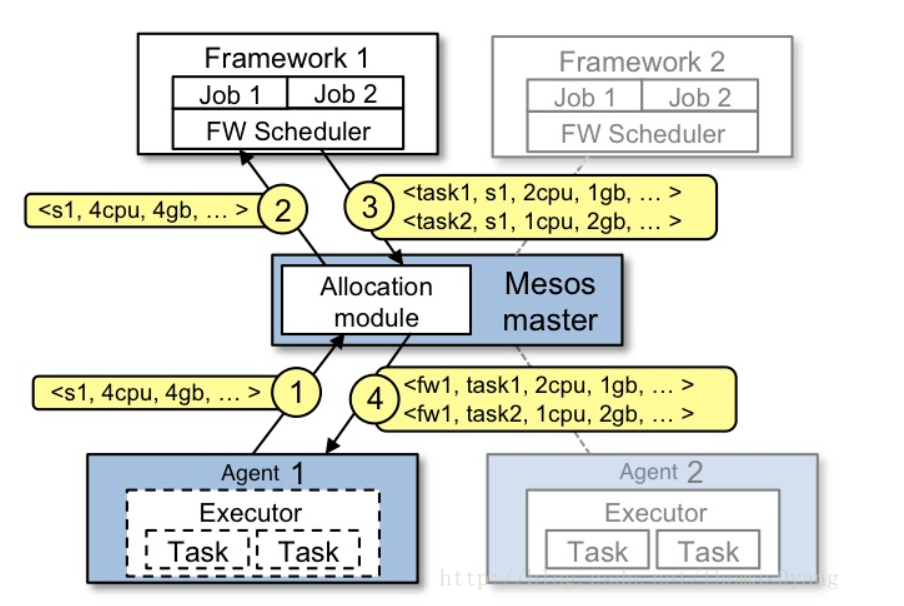
## 过滤器

framework 可以通过过滤器机制告诉 master 它的资源偏好，比如希望分配过来的 offer 中节点上至少空闲多少资源，或者只接受某些节点。主要是为了加速资源分配的收敛。

## 回收机制

master 可以通过回收计算节点上的任务来动态调整长期任务和短期任务的分布。如果某个 framework 在超时内没有为分配的资源返回要运行的任务，则回收该资源。

资源供给的一个例子   
下图描述了一个 Framework 如何通过调度来运行一个 Task



事件流程:

1. Slave1 向 Master 报告，有4个CPU和4 GB内存可用
2. Master 发送一个 Resource Offer 给Framework1 来描述 Slave1 有多少可用资源
3. FrameWork1 中的 FW Scheduler会答复Master，我有两个 Task 需要运行在 Slave1，一个 Task 需要<2个cpu，1gb内存=”“>，另外一个Task需要<1个cpu，2 gb内存=”“
4. Master 发送这些 Tasks 给Slave1。然后，Slave1还有1个CPU和1 GB内存没有使用，所以分配模块可以把这些资源提供给 Framework2

当 Tasks 完成和有新的空闲资源时，Resource Offer 会不断重复这一个过程。

## 任务的启动

**任务的启动过程，就是任务实例和zookeeper进行交互的过程。每个实例在启动过程中，会把自身的信息注册到zookeeper中去。并完成选举和分片策略的设置，也就是完成上文一些zookeeper节点的创建和持久化。**

整个启动的过程都在JobScheduler.init()方法中完成。其中最重要的方法registerStartUpInfo完成了监听，选举持久化数据，以及设置分片标志位（为了任务执行是主节点进行分片算法）等工作。init()方法中完成了配置和注册之后，相关的参数被传递给了JobScheduleController类，这个类就是quartz的封装。之前配置的任务执行类和cron表达式被转换成JobDetail和Trigger这两个quartz的类，然后通过quartz的scheduler.start触发任务。等待任务的执行。

# 一、elastic-job是什么

elastic-job是当当开发的基于qutarz以及zookeeper封装的作业调度工具，主要有两个大框架，一个是elastic-job lite另外一个是elastic-job cloud，其中qutarz是一个开源的作业调度工具，zookeeper是分布式调度工具，这两者结合搭建了elastic-job-lite，这是一个无中心节点的调度，而elastic-job-cloud是一个有中心节点的分布式调度开源工具，只需要设置好机器以及分片，就可以自动的调度到对应的机器上运行，与lite的不同时cloud采用了mesos来进行分布式资源管理，简单的来说两者的不同是：同一个作业在两台机器上跑，lite需要手动在两台机器上跑，但是cloud只需要上传作业包，就可以自动的在两台机器上跑，因为lite不支持作业的调度，为无中心的。

Elastic-Job是一个分布式调度解决方案，由两个相互独立的子项目Elastic-Job-Lite和Elastic-Job-Cloud组成。

Elastic-Job-Lite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采用自研Mesos Framework的解决方案，额外提供资源治理、应用分发以及进程隔离等功能。

Elastic-Job-Cloud 基于 Mesos 实现分布式作业调度，或者说 Elastic-Job-Cloud 是 Mesos 上的 框架( Framework )。

**2. 作业执行类型**

在 Elastic-Job-Cloud，作业执行分成两种类型：

常驻作业

常驻作业是作业一旦启动，无论运行与否均占用系统资源；  
常驻作业适合初始化时间长、触发间隔短、实时性要求高的作业，要求资源配备充足。

瞬时作业

瞬时作业是在作业启动时占用资源，运行完成后释放资源。  
瞬时作业适合初始化时间短、触发间隔长、允许延迟的作业，一般用于资源不太充分，或作业要求的资源多，适合资源错峰使用的场景。

if (CloudJobExecutionType.**TRANSIENT** == jobConfig.getJobExecutionType()) { // 瞬时作业

transientProducerScheduler.register(jobConfig);

} else if (CloudJobExecutionType.DAEMON ==jobConfig.getJobExecutionType()) { //常驻作业

readyService.addDaemon(jobConfig.getJobName());

}

## 3.2 瞬时作业

瞬时作业在调度时，使用**发布瞬时作业任务的调度器**( TransientProducerScheduler )调度作业。当瞬时作业到达作业执行时间，添加到待执行作业队列。

作业APP指作业打包部署后的应用，描述了作业启动需要用到的CPU、内存、启动脚本及应用下载路径等基本信息，每个APP可以包含一个或多个作业。

简单来说，一个云作业App可以理解成由多个作业打在一起的 jar。

FROM <http://dockone.io/article/636>  
Fenzo是一个在Mesos框架上应用的通用任务调度器。它可以让你通过实现各种优化策略的插件，来优化任务调度，同时这也有利于集群的自动缩放。

**Q11：你好，看了看这个框架想请教几个问题：**

1.这个框架是否自带日志搜集模块？  
2.这个框架能否进行性能统计？  
3.这个框架在某个节点资源耗尽时可否自动切换？如果所有节点资源耗尽是否容易崩溃，自恢复能力如何？  
4.这个框架可否配置负载均衡？

1. 带日志模块，但是功能比较简单，没有一个全局的展示
2. 可以进行性能统计
3. Mesos 是根据当前的集群资源统计来决定给调度器分配多少资源的，资源耗尽只会导致新的应用无法部署，不会影响正在运行的东西。
4. 可以配置负载均衡。 并且 Mesos 本身也有多 Master 机制

**Q12：请问 Mesos 怎样决定分配多少资源？分配的资源什么时候回收？**

A12：**Mesos 与其它的集群管理工具不同， Mesos 本身不负责分配资源，它只是将当前集群的剩余资源提供给注册到它的调度器，由调度器本身来决定使用多少资源，以及合适释放资源。**

新Job调度平台show相关的几个Job测试完成，本周在qa环境观察一周，如果顺利，预计于本周日上线~~

为应对新Job上线后带来的风险，我总结了可能发生的几个问题以及应对方案，大家可以看一下

1. 整个Job平台宕机如何处理？

    1) 查看mesos进程，确认是master还是agent进程没了，或者都不存在

         2) 查看elastic-job进程，确认进程是否存在

    3) 启动不存在的mesos进程，启动步骤：先启动master，后启动agent

    4) 启动elastic-job

2. 如果整个Job平台重启无法启动，该如何处理？

    1) kill所有mesos进程，包含master和agent

    2) 远程连接zookeeper，删除mesos节点，比如，dev环境节点是dev\_mesos，prod环境节点是pro\_mesos，**切记不要删除elastic job相关节点**

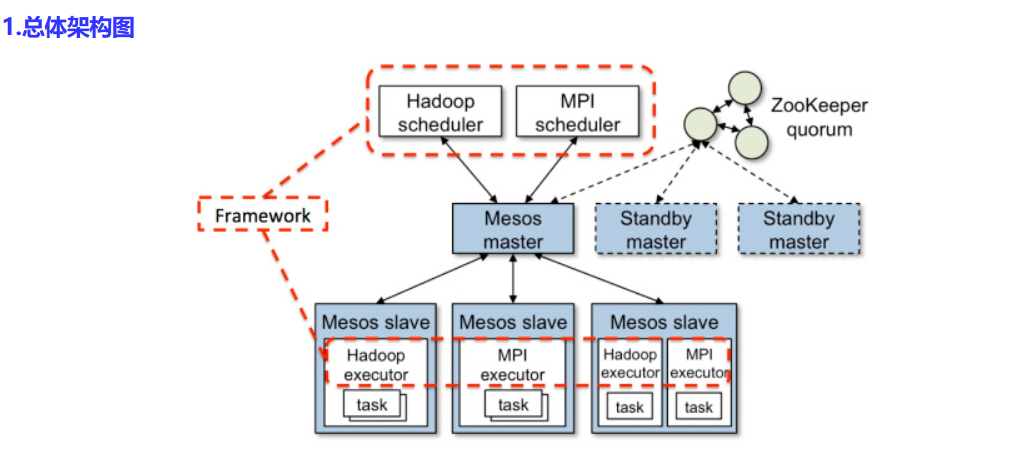
    3) 启动mesos进程，启动步骤：先启动master，后启动agent

   4) 启动elastic-job

2. 某个或某几个Job无法正常执行，如何处理？

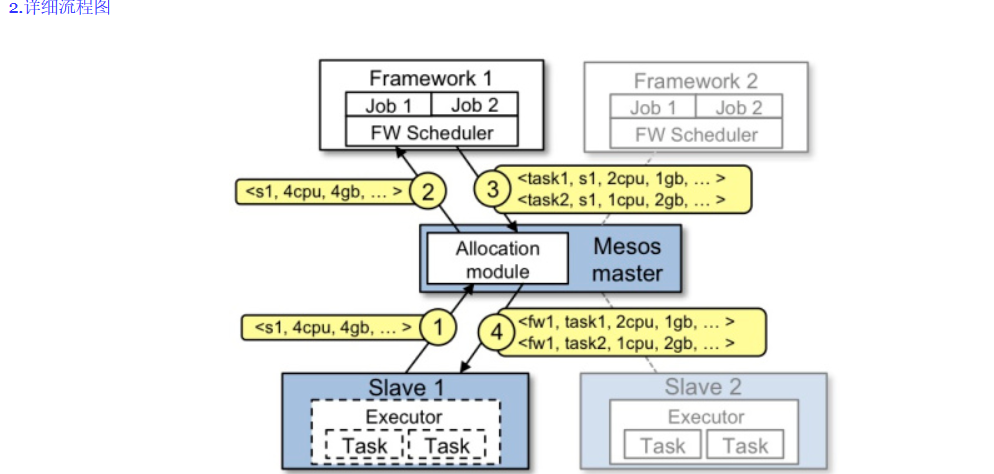
    1) 在mesos上查看Job自身log，定位问题解决

###### Mesos架构理解



上图修改自Apache Mesos网站上的图片，如图所示，Mesos实现了两级调度架构，它可以管理多种类型的应用程序。第一级调度是Master的守护进程，管理Mesos集群中所有节点上运行的Slave守护进程。集群由物理服务器或虚拟服务器组成，用于运行应用程序的任务，比如Hadoop和MPI作业。第二级调度由被称作Framework的“组件”组成。Framework包括调度器（Scheduler）和执行器（Executor）进程，其中每个节点上都会运行执行器。Mesos能和不同类型的Framework通信，每种Framework由相应的应用集群管理。上图中只展示了Hadoop和MPI两种类型，其它类型的应用程序也有相应的Framework。

Mesos Master协调全部的Slave，并确定每个节点的可用资源，  
聚合计算跨节点的所有可用资源的报告，然后向注册到Master的Framework（作为Master的客户端）发出资源邀约。Framework可以根据应用程序的需求，选择接受或拒绝来自master的资源邀约。一旦接受邀约，Master即协调Framework和Slave，调度参与节点上任务，并在容器中执行，以使多种类型的任务，比如Hadoop和Cassandra，可以在同一个节点上同时运行。

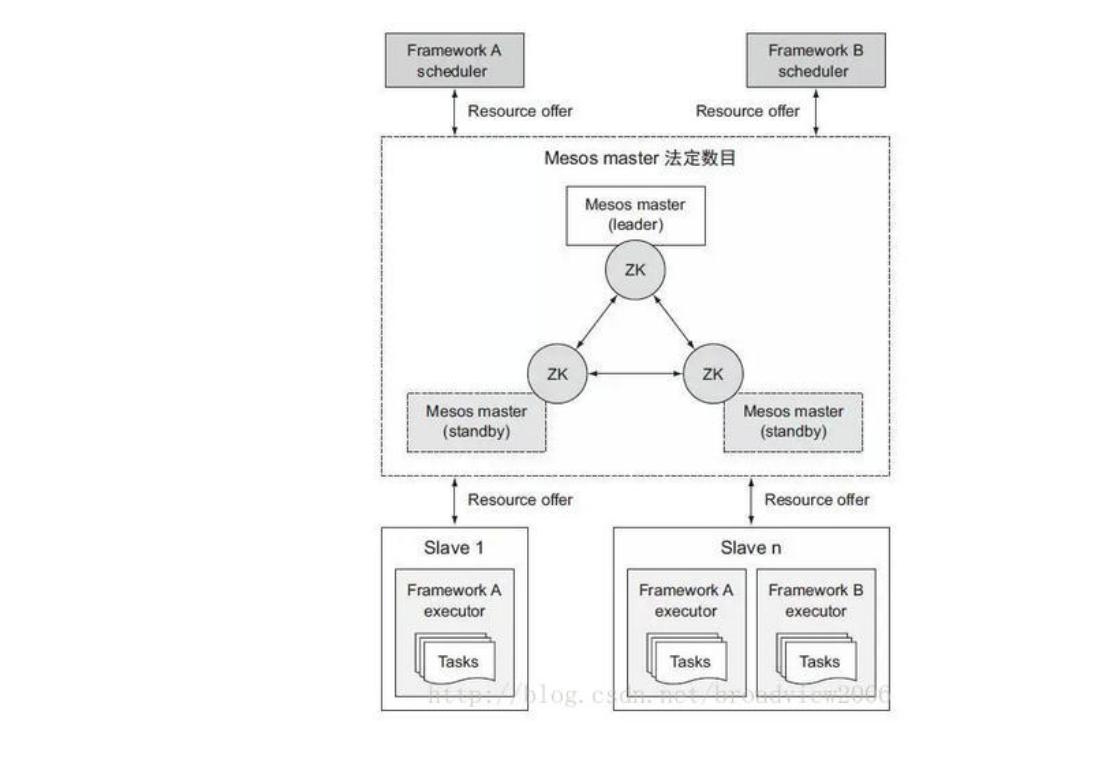


我们来研究下上图的事件流程。上一篇谈到，Slave是运行在物理或虚拟服务器上的Mesos守护进程，是Mesos集群的一部分。Framework由调度器（Scheduler）应用程序和任务执行器（Executor）组成，被注册到Mesos以使用Mesos集群中的资源。

* Slave 1向Master汇报其空闲资源：4个CPU、4GB内存。然后，Master触发分配策略模块，得到的反馈是Framework 1要请求全部可用资源。
* Master向Framework 1发送资源邀约，描述了Slave 1上的可用资源。
* Framework的调度器（Scheduler）响应Master，需要在Slave上运行两个任务，第一个任务分配<2 CPUs, 1 GB RAM>资源，第二个任务分配<1 CPUs, 2 GB RAM>资源。
* 最后，Master向Slave下发任务，分配适当的资源给Framework的任务执行器（Executor）,接下来由执行器启动这两个任务（如图中虚线框所示）。 此时，还有1个CPU和1GB的RAM尚未分配，因此分配模块可以将这些资源供给Framework 2。

**即由Master向注册其上的Framework发送资源邀约。 每次资源邀约包含一份Slave节点上可用的CPU、RAM等资源的列表。 Master提供这些资源给它的Framework，是基于分配策略的。**

首先，Mesos决定使用热备份（hot-standby）设计来实现Master节点集合。正如Tomas Barton对上图的说明，一个Master节点与多个备用（standby）节点运行在同一集群中，并由开源软件Zookeeper来监控。Zookeeper会监控Master集群中所有的节点，并在Master节点发生故障时管理新Master的选举。建议的节点总数是5个，实际上，生产环境至少需要3个Master节点。 Mesos决定将Master设计为持有软件状态，这意味着当Master节点发生故障时，其状态可以很快地在新选举的Master节点上重建。 Mesos的状态信息实际上驻留在Framework调度器和Slave节点集合之中。当一个新的Master当选后，Zookeeper会通知Framework和选举后的Slave节点集合，以便使其在新的Master上注册。彼时，新的 Master可以根据Framework和Slave节点集合发送过来的信息，重建内部状态。

**https://blog.csdn.net/broadview2006/article/details/80129745**

Mesos提供了一套分布式、容错性架构来完成资源的细粒度分配。这套架构包括三个组件：master、slave及运行在其上的应用本身（通常称为framework）。Mesos依赖于Apache ZooKeeper，一个分布式的数据存储系统，专用于集群内的协调同步leader投票选举，以及Mesos master、slave和framework间的leader发现。   
　　在下图中，你能够看见这些架构组件如何在一起工作，从而提供一个稳定的平台来部署应用

# masters

　　Mesos master的职责是管理集群中在每台机器上运行的Mesos slave守护进程。通过ZooKeeper和master之间协调哪个节点是主master，哪些节点作为备用存在，它们将在主master离线时接管服务。   
　　主master节点使用可插拔的分配模块或调度算法来分发资源供给至各种调度器，从而决定将什么资源提供给某一特定的framework。调度器依据其上是否有任务需要执行来决定接收或拒绝资源供给。   
　　Mesos集群至少要求有一个master节点。在生产环境为了保证高可用性，推荐采用三个甚至更多的master节点。你可以将ZooKeeper在与master相同的机器上运行，或者使用独立ZooKeeper集群。

# slaves

在集群中负责执行framework任务的服务器被称为Mesos slave节点，它们访问ZooKeeper来确定主master节点，将CPU、内存、存储资源以资源供给的形式宣告给主master。当调度器从主master接收资源供给后，在slave节点上启动一个或多个执行器，执行器负责运行framework的任务。   
　　Mesos slave也能够基于属性与资源进行配置，从而允许它们定制特定环境。属性配置是键值对形式，可以包含类似于节点所在机房位置信息。资源配置可以替代Mesos自动探测发现slave节点的有效资源，并由用户指定具体的CPU、内存、磁盘资源信息。属性配置与资源配置的示例信息如下：

Mesos是什么

Mesos是一个集群管理平台。 可以理解为是一种分布式系统的kernel， 负责集群资源的分配， 这里的资源指的是CPU资源， 内存资源， 存储资源， 网络资源等。 在Mesos可以运行Spark， Storm， Hadoop， Marathon等多种Framework（框架）。

**Mesos的架构主**要有Masters（主节点）， Slaves（从节点）， 和 及在Mesos上运行的Framework（框架）组成。 各个部分的分工如下：

**Master**： 负责处理Slave节点和Framework间的资源通讯， 根据指定的策略来决定分配多少资源给framework。

**Slave**:  启动本地进程， 同时向Master报告有哪些资源可用。

**Framework**： 接收来自Master提供的Slave节点的资源（如CPU和内存）， Framework由调度器（负责监控和管理Slave的状态）和执行器（负责在服务器执行应用程序代码）组成

* Mesos-master：协调全部的slave，并确定每个节点的可用资源，聚合计算跨节点的所有可用资源的报告，然后向注册到Master的Framework发出资源邀约。
* Mesos-slave：向master汇报自己的空闲资源和任务的状态，负责管理本节点上的各个mesos-task，在framework成功向Master申请资源后,收到消息的slave会启动相应framework的exexutor
* Framework：Hadoop，Spark等，通过MesosSchedulerDiver接入Mesos
* Executor：执行器，用于启动计算框架中的task
* 总体上看，Mesos是一个master/slave结构，其中，master是非常轻量级的，仅保存了framework和mesos slave的一些状态，而这些状态很容易通过framework和slave重新注册而重构，因而很容易使用了zookeeper解决mesos master的单点故障问题。
* Mesos master实际上是一个全局资源调度器，采用某种策略将某个slave上的空闲资源分配给某一个framework，各种framework通过自己的调度器向Mesos master注册，以接入到Mesos中。而Mesos slave主要功能是汇报任务的状态和启动各个framework的executor

如上图所示,Mesos由始至终只做一件事情，就是分布式集群资源的分配。任务的调度和执行由每个计算框架(Framework)自己完成,这样可以容易的实现mesos的扩展性和稳定性。

1. **Slave1向Master汇报其有<4CPU,4GB RAM>的空闲资源，然后Master调用分配模块，告诉Framework1所有可用的空闲资源。**
2. **Master发送一个描述Slave1当前空闲资源的resource offer给框架1。**
3. **Framework1的调度器回复Master，需要运行两个task在Slave1上，第一个task需要资源<2CPU, 1GBRAM>，第二个task需要资源<1CPU, 2GB RAM>。**
4. **Master把任务需求资源发送给Slave1，Slave1分配适当的资源给Framework1的executor，然后executor开始执行这两个任务,因为Slave1还剩<1CPU,1G RAM>的资源还未分配，分配模块可以将这些资源提供给Framwork2来使用。**

每当有task结束，容器会被”销毁”,释放新的资源，都会执行资源邀约(resource offer)进程

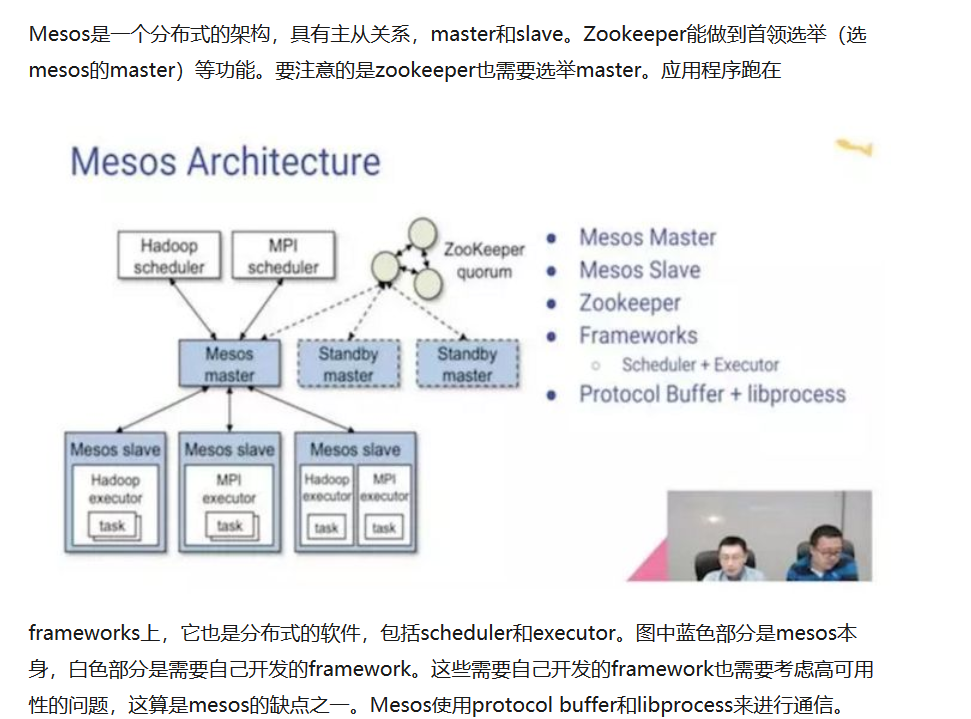
https://segmentfault.com/a/1190000007723430

### Mesos Framework

Mesos的定位是一些资源的调度，它把任务的调度交给了Framework来做，Mesos只关心资源以及把资源给了谁， Framework来决定哪些资源怎么去使用。Mesos鼓励Framework在上面共生。想象一下，作为一个大型公司，有很多的资源，有核心的一组人来维护Mesos的集群，不断的往Mesos上添加资源和减少资源，而把Framework执行的能力交给其它的组、需要资源的那些组，各个组就可以写自己的Framework，丢到整个大的Mesos集群上来执行了。**Mesos框架上和执行上各种各样的Framework，而Mesos本身也不了解为什么Framework工作，它只是知道把Offer给Framework，然后Framework告诉它来执行什么样的Executor。**

从前面提到的[两级架构的说明](http://www.infoq.com/cn/articles/analyse-mesos-part-02)一文中我们知道，Mesos Master代理任务的调度首先从Slave节点收集有关可用资源的信息，然后以资源邀约的形式，将这些资源提供给注册其上的Framework。

Framework可以根据是否符合任务对资源的约束，选择接受或拒绝资源邀约。一旦资源邀约被接受，Framework将与Master协作调度任务，并在数据中心的相应Slave节点上运行任务。

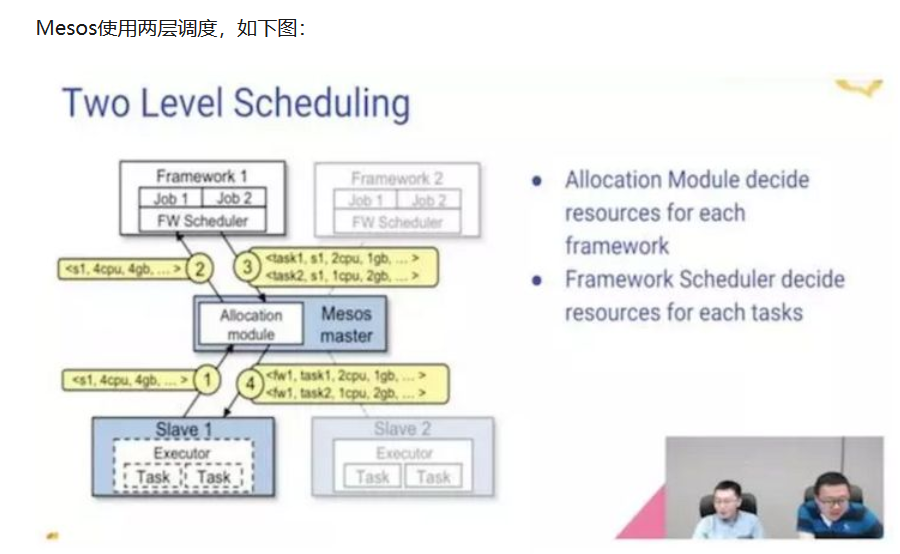


1）zookeeper做为leader election。每个master起来的时候都要向zookeeper的quorum注册znode。

2）master主要用于管理slave node的状态，知道slave在哪儿，有哪些资源，同时知道各个框架的信息。比如Hadoop和MPI框架的基本信息。master还可以发resource offer。如果master down了，zookeeper可以马上从standby master里面选一个。因为几个master里面共享的东西很少，不需要数据的同步。

3）slave负责运行任务，同时不断的向master发消息，报告可用资源和在运行的任务的状态。这两种消息并不相同，报告任务状态的消息保证一定delivery，而其他消息并不保证。

4）framwork是一个分布式的应用，包括scheduler和executor。其中scheduler部分要和master紧密通信，接受master发送的offer，scheduler收到offer后决定是不是要跑某个task。executor真正的去执行任务。Offer的发送由master决定。



Mesos master里面有Application module，负责具体的分配算法。这个模块本身有默认算法设置，也可以自己实现。

首先slave向master报告自己的资源信息，比如4个CPU等等，在application module计算之后发给框架1，框架1里面有task queue，比如图中有两个Job，scheduler分配给不同task不同的资源，把信息发给master，master再返还给对应的slave去跑task。这就是two level scheduling，one level scheduling是指slave直接和scheduler通信。

**定义一个最小化的接口来支持跨框架的资源共享，其他的调度以及执行工作都委托给框架自己来控制。**

这句话标志Mesos不试图作为一个一栈式解决问题的系统，而是以最小的成本来实现资源共享。先来看看官方提供的架构图：

主要组件以及概念：

* Zookeeper 主要用来实现Master的选举，支持Master的高可用。
* Master Mesos的主节点，接收Slave和Framework scheduler的注册，分配资源。
* Slave 从节点，接收master发来的task，调度executor去执行。
* Framework 比如上图中的Hadoop，MPI就是Framework，包括scheduler，executor两部分。**scheduler独立运行，启动后注册到master，接收master发送的Resource Offer消息**，来决定是否接受。Executor是给slave调用的，执行framework的task。Mesos内置了CommandExecutor（直接调用shell）和DockerExecutor两种executor，其他的自定义Executor需要提供uri，供slave下载。
* **Task Mesos最主要的工作其实就是分配资源，然后询问scheduler是否可以利用该资源执行Task，scheduler将资源和Task绑定后交由Master发送给指定的Slave执行。Task可以是长生命周期的，也可以使用批量的短生命周期的。**

官方提供的另外一个资源分配的例子：

1. Slave1 向 Master 报告，有4个CPU和4 GB内存可用
2. Master 发送一个 Resource Offer 给 Framework1 来描述 Slave1 有多少可用资源
3. FrameWork1 中的 FW Scheduler会答复 Master，我有两个 Task 需要运行在 Slave1，一个 Task 需要<2个CPU，1 GB内存=””>，另外一个Task需要<1个CPU，2 GB内存=””>
4. 最后，Master 发送这些 Tasks 给 Slave1。然后，Slave1还有1个CPU和1 GB内存没有使用，所以分配模块可以把这些资源提供给 Framework2

这个例子可以看出来，Mesos的核心工作其实很少，资源管理和分配以及task转发。调度由Framework实现，task的定义以及具体执行也由Framework实现，Mesos的资源分配粒度是按task的，但由于executor执行task可能在同一个进程中实现，所以资源限制只是一种流控的机制，并不能实际的控制到task这个粒度。

看了上面官方的架构说明，大约应该明白了Mesos的大致架构，但具体Mesos为什么要做成这样，下面我们具体分析下。

**Detector模块用来感知当前的master是谁，它主要利用ZooKeeper中的watcher的机制来监控选举的Group（/mesos目录下的子节点）的变化。**ZooKeeper提供了getChildren()应用程序接口，此接口可以用来监控一个目录下子节点的变化，如果一个新子节点加入或者原来的节点被删除，那么这个函数调用会立即返回当前目录下的所有节点，然后Detector模块可以挑选序号最小的作为master节点。

每一个Mesos master会同时使用Contender和Detector模块，用Contender进行master竞选。在master节点上使用Detector模块的原因是在Mesos的高可用集群中，你可以使用任意一个master节点的地址和端口来访问Mesos的WebUI，当访问一个Replica节点时，Mesos会把这个浏览器的链接请求自动转发到通过Detector模块探测到的master节点上。

其他的Mesos组件，例如Mesos Agent，Framework scheduler driver会使用Detector模块来获取当前的Mesos master，然后向它注册，当master发送变化时，Detecor模块会第一时间通知，它们会重新注册（re-register)。

由于IBM在Mesos社区的推动，在MESOS-4610项目中，Mesos Contender和Detector已经可以支持 以插件的方式进行加载。现在Mesos社区官方仅支持用Zookeeper集群进行Leader选举，在支持了插件的方式加载后，用户可以实现自己的插件，用另外的方式比如选择用etcd（MESOS-1806）、consule（MESOS-3797）等集群进行Leader选举。

注意：现在Mesos仅用ZooKeeper进行Leader的选举，并没有用它进行数据的共享。在Mesos中有一个Replicated Log模块，负责进行多个master之间的数据共享、同步等。可以参考Mesos的官方文档获取详细的设计<http://mesos.apache.org/documentation/latest/replicated-log-internals/>。同时为了使Mesos的高可用不依赖与一个第三方的集群，现在社区正在考虑用Replicated log替代第三方集群进行Leader选举，具体进度可以参考MESOS-3574项目。

**http://dockone.io/question/968**

而Mesos，根据我查阅论文和网络上的相关资料，各个计算框架并不向master去申请资源，而是mesos master主动将资源给各个框架（resource offer），然后框架根据需求决定是否接受分配的资源，如果接受了资源，再由自己的调度模块去将资源分配给各个任务。  
  
那么问题是：如果framework不向mesos申请资源，mesos master根据什么去给各个框架分配资源呢？目前主要采用的资源分配策略是DRF，mesos master如何知道各个框架的主资源是什么？

<http://developer.51cto.com/art/201401/426500.htm>

<http://developer.51cto.com/art/201401/426462.htm>

<http://dongxicheng.org/apache-mesos/mesos-scheduler/>

# [elastic-job详解（一）：数据分片](https://www.cnblogs.com/haoxinyue/p/6919375.html)

数据分片的目的在于把一个任务分散到不同的机器上运行，既可以解决单机计算能力上限的问题，也能降低部分任务失败对整体系统的影响。elastic-job并不直接提供数据处理的功能，框架只会将分片项分配至各个运行中的作业服务器（其实是Job实例，部署在一台机器上的多个Job实例也能分片），开发者需要自行处理分片项与真实数据的对应关系。框架也预置了一些分片策略：平均分配算法策略，作业名哈希值奇偶数算法策略，轮转分片策略。同时也提供了自定义分片策略的接口。

## 分片原理

elastic-job的分片是通过zookeeper来实现的。分片的分片由主节点分配，如下三种情况都会触发主节点上的分片算法执行：

* 新的Job实例加入集群
* 现有的Job实例下线（如果下线的是leader节点，那么先选举然后触发分片算法的执行）
* 主节点选举

上述三种情况，会让zookeeper上leader节点的sharding节点上多出来一个necessary的临时节点，主节点每次执行Job前，都会去看一下这个节点，如果有则执行分片算法。

**1. 基于平均分配算法的分片策略**

基于平均分配算法的分片策略对应的类是：AverageAllocationJobShardingStrategy。它是默认的分片策略。它的分片效果如下：

* 如果有3个Job实例, 分成9片, 则每个Job实例分到的分片是: 1=[0,1,2], 2=[3,4,5], 3=[6,7,8].
* 如果有3个Job实例, 分成8片, 则每个Job实例分到的分片是: 1=[0,1,6], 2=[2,3,7], 3=[4,5].
* 如果有3个Job实例, 分成10片, 则个Job实例分到的分片是: 1=[0,1,2,9], 2=[3,4,5], 3=[6,7,8].

**2. 作业名的哈希值奇偶数决定IP升降序算法的分片策略**

这个策略的对应的类是：OdevitySortByNameJobShardingStrategy，它内部其实也是使用AverageAllocationJobShardingStrategy实现，只是在传入的节点实例顺序不一样，也就是上面接口参数的List<JobInstance>。AverageAllocationJobShardingStrategy的缺点是一旦分片数小于Job实例数，作业将永远分配至IP地址靠前的Job实例上，导致IP地址靠后的Job实例空闲。而OdevitySortByNameJobShardingStrategy则可以根据作业名称重新分配Job实例负载。如：

* 如果有3个Job实例，分成2片，作业名称的哈希值为奇数，则每个Job实例分到的分片是：1=[0], 2=[1], 3=[]
* 如果有3个Job实例，分成2片，作业名称的哈希值为偶数，则每个Job实例分到的分片是：3=[0], 2=[1], 1=[]

## 1. 前言

调度器是Mesos的核心部件，主要负责将各个slave上资源分配给各个framework，常见的调度机制有FIFO，Fair Scheduler，Capacity Scheduler，Quincy，Condor等。Mesos为了支持多framework接入，采用了双层调度机制，首先，由mesos中的allocator将资源分配给framework，然后又由framework自己的调度器将资源分配给任务。本文重点介绍mesos中的allocator模块。

（**什么是apache mesos？**参考：《[统一资源管理与调度平台（系统）介绍](http://dongxicheng.org/mapreduce-nextgen/mesos_vs_yarn/)》,本文分析基于Mesos [SVN Revision 1327410](https://svn.apache.org/repos/asf/incubator/mesos/trunk/)）

## 2. Mesos调度机制

Mesos中的调度机制被称为“Resource Offer”，采用了基于资源量的调度机制，这不同于Hadoop中的基于slot的机制。在mesos中，slave直接将资源量（CPU和内存）汇报给master，由master将资源量按照某种机制分配给framework，其中，“某种机制”是“Dominant Resource Fairness（DRF）”

对于类似mesos采用双层调度框架的系统，在设计时，需要解决以下问题：“Mesos在不知道各个framework资源需求的情况下，如何满足其需求？”，更具体一些，“Mesos在不知道framework中哪些数据存放在哪些节点情况下，如何做到数据locality？”为了解决该问题，mesos提供了**“reject offer”**机制，允许framework暂时拒绝不满足其资源需求的slave，在此，mesos采用了类似于Hadoop中的“delay scheduling“调度机制。

在mesos中，作业调度是一个分布式的过程，当出现失败情况时，需要表现出一定的高效性和鲁棒性。为此，mesos提供了以下机制：

（1）**filters机制**。 每次调度过程，mesos-master需要与framework-scheduler进行通信，如果有些framework总是拒绝slave，那么由于额外的通信开销会使得调度性能低效。为此避免不必要的通信，mesos提供了filters机制，允许framework只接收“剩余资源量大于L的slave”或者“只接收node列表中的slave”。

（2）**rescinds机制**。如果某个framework在一定的时间内没有为分配的资源返回对应的任务，则mesos会回收其资源量，并将这些资源分配给其他framework。

<https://blog.csdn.net/zsw7181084/article/details/54691657>

<https://blog.csdn.net/zsw7181084/article/details/54691657>

之后，向域名的5050发出get请求，由于mesos这个域名，dns不能解析，所以导致访问失败，所以会一直不停地弹出以上的问题对话框。

经过研究，发现state、snapshot这些请求，会默认把mesos master的hostname作为域名去做get请求。可以在$MESOS\_HOME/etc/mesos/mesos-maste-env.sh中设置hostname的值（MESOS\_hostname）， 如果未设置会从环境中得到真正的hostname

解决办法：

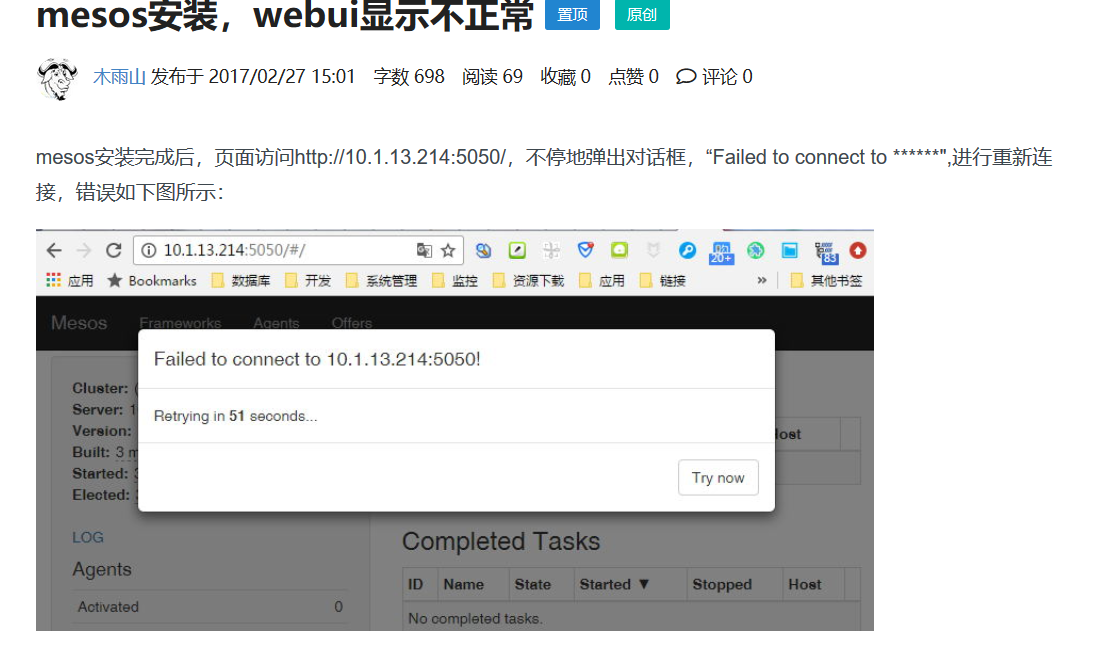
1.将域名的映射添加到浏览器所在机器的hosts里

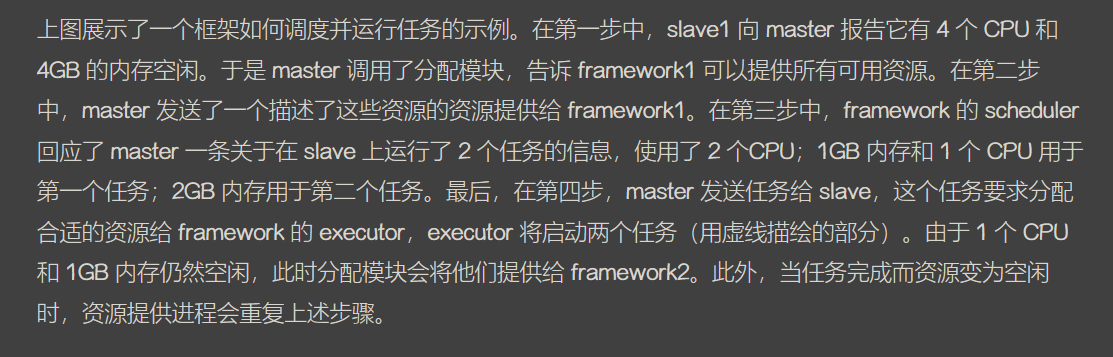
2.如果只是练习，可以直接将 MESOS\_hostname 设置为ip地址，这样就会从ip里访问，从而问题得到解决。

**https://blog.csdn.net/boonya/article/details/51942040**

用上边的启动配置方式，所有的Mesos master节点会通过ZooKeeper进行Leader的选举，所有的Mesos slave节点和Framework都会和ZooKeeper进行通信，获取当前的Mesos master Leader，并且会一直检测Master节点的变化。当Leader故障时，ZooKeeper会第一时间选出新Leader，然后所有的Slave节点和Framework都会获取到新Leader进行重新注册。

<https://my.oschina.net/duxuefeng/blog/847070>





# slaves

　　在集群中负责执行framework任务的服务器被称为Mesos slave节点，它们访问ZooKeeper来确定主master节点，将CPU、内存、存储资源以资源供给的形式宣告给主master。当调度器从主master接收资源供给后，在slave节点上启动一个或多个执行器，执行器负责运行framework的任务。   
　　Mesos slave也能够基于属性与资源进行配置，从而允许它们定制特定环境。属性配置是键值对形式，可以包含类似于节点所在机房位置信息。资源配置可以替代Mesos自动探测发现slave节点的有效资源，并由用户指定具体的CPU、内存、磁盘资源信息。属性配置与资源配置的示例信息如下：

--attributes='datacenter:pdx1;rack:1-1;os:rhel7'

--resources='cpu:24;mem:24576;disk:409600'

总体上看，Mesos是一个master/slave结构，其中，master是非常轻量级的，仅保存了framework和mesos slave的一些状态，而这些状态很容易通过framework和slave重新注册而重构，因而很容易使用了zookeeper解决mesos master的单点故障问题。

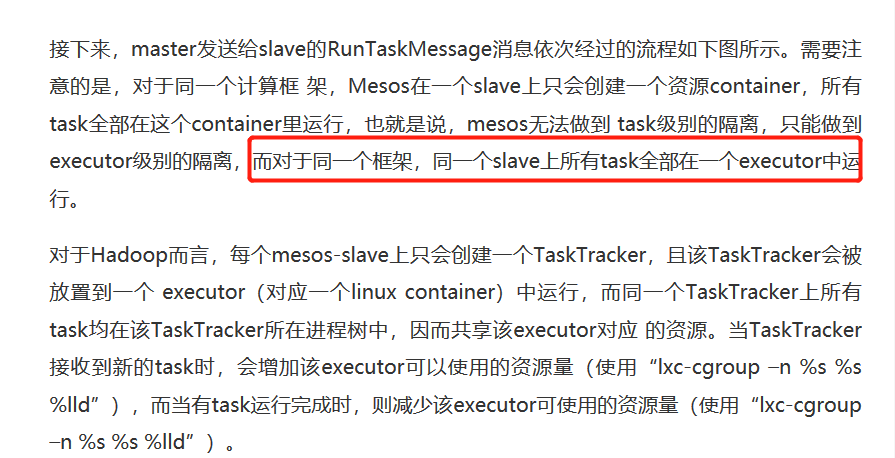
Mesos master实际上是一个全局资源调度器，采用某种策略将某个slave上的空闲资源分配给某一个framework，各种framework通过自己的调度器向Mesos master注册，以接入到Mesos中。而Mesos slave主要功能是汇报任务的状态和启动各个framework的executor

#### 事件流程:

1. Slave1 向 Master 报告，有4个CPU和4 GB内存可用
2. Master 发送一个 Resource Offer 给 Framework1 来描述 Slave1 有多少可用资源
3. FrameWork1 中的 FW Scheduler会答复 Master，我有两个 Task 需要运行在 Slave1，一个 Task 需要<2个CPU，1 GB内存="">，另外一个Task需要<1个CPU，2 GB内存="">
4. 最后，Master 发送这些 Tasks 给 Slave1。然后，Slave1还有1个CPU和1 GB内存没有使用，所以分配模块可以把这些资源提供给 Framework2

当 Tasks 完成和有新的空闲资源时，Resource Offer 会不断重复这一个过程。、

[**http://developer.51cto.com/art/201401/426500.htm**](http://developer.51cto.com/art/201401/426500.htm)



我们分析结果，可以得出结论：当线程只剩下守护线程的时候,JVM就会退出；补充一点如果还有其他的任意一个用户线程还在，JVM就不会退出。

## 使用它需要注意些什么？

1. thread.setDaemon(true)必须在thread.start()之前设置，否则会跑出一个IllegalThreadStateException异常。你不能把正在运行的常规线程设置为守护线程。
2. 在Daemon线程中产生的新线程也是Daemon的。
3. 守护线程不能用于去访问固有资源，比如读写操作或者计算逻辑。因为它会在任何时候甚至在一个操作的中间发生中断。
4. Java自带的多线程框架，比如ExecutorService，会将守护线程转换为用户线程，所以如果要使用后台线程就不能用Java的线程池。

## 意义及应用场景

当主线程结束时，结束其余的子线程（守护线程）自动关闭，就免去了还要继续关闭子线程的麻烦。如：Java垃圾回收线程就是一个典型的守护线程；内存资源或者线程的管理，但是非守护线程也可以。

它的存在，必定有它的意义，只需在乎我们怎么把它用到恰到好处。