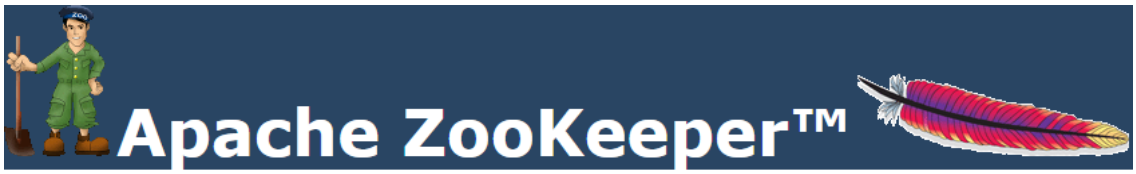
cyfonly
若无力驾驭，自由便是负担

构建高可用ZooKeeper集群

ZooKeeper 是 Apache 的一个顶级项目，为分布式应用提供高效、高可用的分布式协调服务，提供了诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知和分布式锁等分布式基础服务。由于 ZooKeeper 便捷的使用方式、卓越的性能和良好的稳定性，被广泛地应用于诸如 Hadoop、HBase、Kafka 和 Dubbo 等大型分布式系统中。

> 本文的目标读者是对 ZooKeeper 有一定了解的技术人员，将从 ZooKeeper 运行模式、集群组成、容灾和水平扩容四方面逐步深入，最终构建出高可用的 ZooKeeper 集群。



一、运行模式

Zookeeper 有三种运行模式：单机模式、伪集群模式和集群模式。

1.1 单机模式

这种模式一般适用于开发测试环境，一方面我们没有那么多机器资源，另外就是平时的开发调试并不需要极好的稳定性。

在 Linux 环境下运行单机模式需要执行以下步骤：

1. 准备 Java 运行环境

安装 Java 1.6 或更高版本的 JDK，并配置好 Java 相关的环境变量 $JAVA_HOME 。

2. 下载 ZooKeeper 安装包

下载地址：http://zookeeper.apache.org/releases.html。选择最新的 stable 版本并解压到指定目录，我们用 $ZK_HOME 表示该目录。

3. 配置 zoo.cfg

首次使用 ZooKeeper，需要将 $ZK_HOME 下的 zoo_sample.cfg 文件重命名为 zoo.cfg，并进行以下配置

```
tickTime=2000     ##Zookeeper最小时间单元，单位毫秒(ms)，默认值为3000
dataDir=/var/lib/zookeeper     ##Zookeeper服务器存储快照文件的目录，必须配置
dataLogDir=/var/lib/log      ##Zookeeper服务器存储事务日志的目录，默认为dataDir
clientPort=2181     ##服务器对外服务端口，一般设置为2181
initLimit=5     ##Leader服务器等待Follower启动并完成数据同步的时间，默认值10，表示tickTime的10倍
syncLimit=2     ##Leader服务器和Follower之间进行心跳检测的最大延时时间，默认值5，表示tickTime的5倍
```

4. 启动服务

使用 $ZK_HOME/bin 目录下的 zkServer.sh 脚本进行服务的启动。

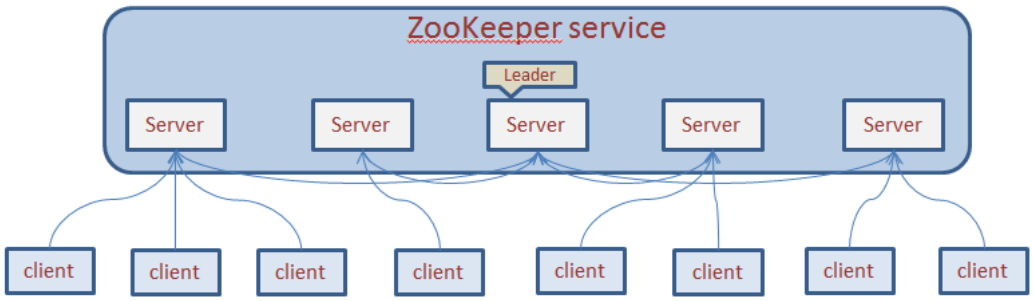
1.2 集群模式

一个 ZooKeeper 集群通常由一组机器组成，一般 3 台以上就可以组成一个可用的 ZooKeeper 集群了。

组成 ZooKeeper 集群的每台机器都会在内存中维护当前的服务器状态，并且每台机器之间都会互相保持通信。

> 重要的一点是，只要集群中存在超过一半的机器能够正常工作，那么整个集群就能够正常对外服务。

ZooKeeper 的客户端程序会选择和集群中的任意一台服务器创建一个 TCP 连接，而且一旦客户端和服务器断开连接，客户端就会自动连接到集群中的其他服务器。



那么如何运行 ZooKeeper 集群模式呢？首先假如我们有三台服务器，IP 分别为 IP_1、IP_2 和 IP_3，则需要执行以下步骤：

1. 准备 **Java** 运行环境（同上）

2. 下载 **ZooKeeper** 安装包（同上）

3. 配置 **zoo.cfg**

```
tickTime=2000
dataDir=/var/lib/zookeeper
dataLogDir=/var/lib/log
clientPort=2181
initLimit=5
syncLimit=2
server.1=IP1:2888:3888
server.2=IP2:2888:3888
server.3=IP3:2888:3888
```

> 可以看到，相比于单机模式，集群模式多了 server.id=host:$port_1$:$port_2$ 的配置。

其中，id 被称为 Server ID，用来标识该机器在集群中的机器序号（在每台机器的 dataDir 目录下创建 myid 文件，文件内容即为该机器对应的 Server ID 数字）。host 为机器 IP，$port_1$ 用于指定 Follower 服务器与 Leader 服务器进行通信和数据同步的端口，$port_2$ 用于进行 Leader 选举过程中的投票通信。

4. 创建 **myid** 文件

在 dataDir 目录下创建名为 myid 的文件，在文件第一行写上对应的 Server ID。

5. 按照相同步骤，为其他机器配置 **zoo.cfg** 和 **myid**文件

6. 启动服务

1.3 伪集群模式

这是一种特殊的集群模式，即集群的所有服务器都部署在一台机器上。当你手头上有一台比较好的机器，如果作为单机模式进行部署，就会浪费资源，这种情况下，ZooKeeper允许你在一台机器上通过启动不同的端口来启动多个 ZooKeeper 服务实例，以此来以集群的特性来对外服务。

这种模式下，只需要把 zoo.cfg 做如下修改：

```
tickTime=2000
dataDir=/var/lib/zookeeper
dataLogDir=/var/lib/log
clientPort=2181
initLimit=5
syncLimit=2
server.1=IP1:2888:3888
server.2=IP1:2889:3889
server.3=IP1:2890:3890
```

二、集群组成

要搭建一个高可用的 ZooKeeper 集群，我们首先需要确定好集群的规模。

> 关于 ZooKeeper 集群的服务器组成，相信很多对 ZooKeeper 了解但是理解不够深入的读者，都存在或曾经存在过这样一个错误的认识：为了使得 ZooKeeper 集群能够顺利地选举出 Leader，必须将 ZooKeeper 集群的服务器数部署成奇数。这里我们需要澄清的一点是：任意台 ZooKeeper 服务器都能部署且能正常运行。

那么存在于这么多读者中的这个错误认识是怎么回事呢？其实关于 ZooKeeper 集群服务器数，ZooKeeper 官方确实给出了关于奇数的建议，但绝大部分 ZooKeeper 用户对于这个建议认识有偏差。在本书前面提到的“**过半存活即可用**”特性中，我们已经了解了，一个 ZooKeeper 集群如果要对外提供可用的服务，那么集群中必须要有过半的机器正常工作并且彼此之间能够正常通信。基于这个特性，如果想搭建一个能够允许 N 台机器 down 掉的集群，那么就要部署一个由 2*N+1 台服务器构成的 ZooKeeper 集群。因此，一个由 3 台机器构成的 ZooKeeper 集群，能够在挂掉 1 台机器后依然正常工作，而对于一个由 5 台服务器构成的 ZooKeeper 集群，能够对 2 台机器挂掉的情况进行容灾。注意，如果是一个由6台服务器构成的 ZooKeeper 集群，同样只能够挂掉 2 台机器，因为如果挂掉 3 台，剩下的机器就无法实现过半了。

> 因此，从上面的讲解中，我们其实可以看出，对于一个由 6 台机器构成的 ZooKeeper 集群来说，和一个由 5 台机器构成的 ZooKeeper 集群，其在容灾能力上并没有任何显著的优势，反而多占用了一个服务器资源。基于这个原因，ZooKeeper 集群通常设计部署成奇数台服务器即可。

三、容灾

所谓容灾，在 IT 行业通常是指我们的计算机信息系统具有的一种在遭受诸如火灾、地震、断电和其他基础网络设备故障等毁灭性灾难的时候，依然能够对外提供可用服务的能力。

对于一些普通的应用，为了达到容灾标准，通常我们会选择在多台机器上进行部署来组成一个集群，这样即使在集群的一台或是若干台机器出现故障的情况下，整个集群依然能够对外提供可用的服务。

而对于一些核心应用，不仅要通过使用多台机器构建集群的方式来提供服务，而且还要将集群中的机器部署在两个机房，这样的话，即使其中一个机房遭遇灾难，依然能够对外提供可用的服务。

上面讲到的都是应用层面的容灾模式，那么对于 ZooKeeper 这种底层组件来说，如何进行容灾呢？讲到这里，可能多少读者会有疑问，ZooKeeper 既然已经解决了单点问题，那为什么还要进行容灾呢？

3.1 单点问题

单点问题是分布式环境中最常见也是最经典的问题之一，在很多分布式系统中都会存在这样的单点问题。

> 具体地说，单点问题是指在一个分布式系统中，如果某一个组件出现故障就会引起整个系统的可用性大大下降甚至是处于瘫痪状态，那么我们就认为该组件存在单点问题。

ZooKeeper 确实已经很好地解决了单点问题。我们已经了解到，基于“过半”设计原则，ZooKeeper 在运行期间，集群中至少有过半的机器保存了最新的数据。因此，只要集群中超过半数的机器还能够正常工作，整个集群就能够对外提供服务。

3.2 容灾

解决了单点问题，是不是该考虑容灾了呢？答案是否定的，在搭建一个高可用的集群的时候依然需要考虑容灾问题。正如上面讲到的，如果集群中超过半数的机器还在正常工作，集群就能够对外提供正常的服务。

> 那么，如果整个机房出现灾难性的事故，这时显然已经不是单点问题的范畴了。

在进行 ZooKeeper 的容灾方案设计过程中，我们要充分考虑到“过半原则”。也就是说，无论发生什么情况，我们必须保证 ZooKeeper 集群中有超过半数的机器能够正常工作。因此，通常有以下两种部署方案。

3.3.1 双机房部署

在进行容灾方案的设计时，我们通常是以机房为单位来考虑问题。在现实中，很多公司的机房规模并不大，因此双机房部署是个比较常见的方案。但是遗憾的是，在目前版本的 ZooKeeper 中，还没有办法能够在双机房条件下实现比较好的容灾效果——因为无论哪个机房发生异常情况，都有可能使得 ZooKeeper 集群中可用的机器无法超过半数。当然，在拥有两个机房的场景下，通常有一个机房是主要机房（一般而言，公司会花费更多的钱去租用一个稳定性更好、设备更可靠的机房，这个机房就是主要机房，而另外一个机房则更加廉价一些）。我们唯一能做的，就是尽量在主要机房部署更多的机器。例如，对于一个由 7 台机器组成的 ZooKeeper 集群，通常在主要机房中部署 4 台机器，剩下的 3 台机器部署到另外一个机房中。

3.3.2 三机房部署

既然在双机房部署模式下并不能实现好的容灾效果，那么对于有条件的公司，选择三机房部署无疑是个更好的选择，**无论哪个机房发生了故障，剩下两个机房的机器数量都超过半数。**假如我们有三个机房可以部署服务，并且这三个机房间的网络状况良好，那么就可以在三个机房中都部署若干个机器来组成一个 ZooKeeper 集群。

我们假定构成 ZooKeeper 集群的机器总数为 N，在三个机房中部署的 ZooKeeper 服务器数分别为 N_1、N_2 和 N_3，如果要使该 ZooKeeper 集群具有较好的容灾能力，我们可以根据如下算法来计算 ZooKeeper 集群的机器部署方案。

1. 计算 N_1

如果 ZooKeeper 集群的服务器总数是 N，那么：

```
N1 = (N-1)/2
```

在 Java 中，"/" 运算符会自动对计算结果向下取整操作。举个例子，如果 N=8，那么 N_1=3；如果 N=7，那么 N_1 也等于 3。

2. 计算 N_2 的可选值

N_2 的计算规则和 N_1 非常类似，只是 N_2 的取值是在一个取值范围内：

```
N2 的取值范围是 1~(N-N1)/2
```

即如果 N=8，那么 N_1=3，则 N_2 的取值范围就是 1~2，分别是 1 和 2。注意，1 和 2 仅仅是 N_2 的可选值，并非最终值——如果 N_2 为某个可选值的时候，无法计算出 N_3 的值，那么该可选值也无效。

3. 计算 N_3，同时确定 N_2 的值

很显然，现在只剩下 N_3 了，可以简单的认为 N_3 的取值就是剩下的机器数，即：

```
N3 = N - N1 - N2
```

只是 N_3 的取值必须满足 $N_3 < N_1+N2$。在满足这个条件的基础下，我们遍历步骤 2 中计算得到的 N_2 的可选值，即可得到三机房部署时每个机房的服务器数量了。

现在我们以 7 台机器为例，来看看如何分配三机房的机器分布。根据算法的步骤 1，我们首先确定 N_1 的取值为 3。根据算法的步骤 2，我们确定了 N_2 的可选值为 1 和 2。最后根据步骤 3，我们遍历 N_2 的可选值，即可得到两种部署方案，分别是 (3,1,3) 和 (3,2,2)。以下是 Java 程序代码对以上算法的一种简单实现：

```
public class Allocation {

    static final int n = 7;
    public static void main(String[] args){
        int n1,n2,n3;
        n1 = (n-1) / 2;
        int n2_max = (n-n1) / 2;
        for(int i=1; i<=n2_max; i++){
            n2 = i;
            n3 = n - n1 -n2;
            if(n3 >= (n1+n2)){
                continue;
            }
            System.out.println("("+n1+","+n2+","+n3+")");
        }
    }
}
```

四、水平扩容

水平可扩容可以说是对一个分布式系统在高可用性方面提出的基本的，也是非常重要的一个要求，通过水平扩容能够帮助系统在不进行或进行极少改进工作的前提下，快速提高系统对外的服务支撑能力。简单地讲，水平扩容就是向集群中添加更多的机器，以提高系统的服务质量。

很遗憾的是，ZooKeeper 在水平扩容扩容方面做得并不十分完美，需要进行整个集群的重启。通常有两种重启方式，一种是集群整体重启，另外一种是逐台进行服务器的重启。

4.1 整体重启

所谓集群整体重启，就是先将整个集群停止，然后更新 ZooKeeper 的配置，然后再次启动。如果在你的系统中，ZooKeeper 并不是个非常核心的组件，并且能够允许短暂的服务停止（通常是几秒钟的时间间隔），那么不妨选择这种方式。在整体重启的过程中，所有该集群的客户端都无法连接上集群。等到集群再次启动，这些客户端就能够自动连接上——注意，整体启动前建立起的客户端会话，并不会因为此次整体重启而失效。也就是说，在整体重启期间花费的时间将不计入会话超时时间的计算中。

4.2 逐台重启

这种方式更适合绝大多数的实际场景。在这种方式中，每次仅仅重启集群中的一台机器，然后逐台对整个集群中的机器进行重启操作。这种方式可以在重启期间依然保证集群对外的正常服务。

参考文章：《从Paxos到Zookeeper 分布式一致性原理与实践》

分类：架构

标签：zookeeper

« 上一篇：mysql数据库开发常见问题及优化
» 下一篇：netty5 HTTP协议栈浅析与实践

评论列表

回复 引用
#1楼 2016-06-30 11:33 tristanTT

感谢分享！

支持(0) 反对(0)

回复 引用
#2楼 2016-06-30 12:31 sonicee

整理的不错，对于ZK小白也能看得懂

支持(0) 反对(0)

回复 引用
#3楼 2016-06-30 12:46 ChokCoco

前排支持 好文要顶。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

发表评论

昵称： Kingson_Wu

评论内容：

提交评论 退出登录 订阅评论

[Ctrl+Enter快捷键提交]

最新IT新闻:
· 2017年软件开发人员需要面对的7个改变
· 腾讯手游业务首次单季度营收超百亿，鹅厂有王者荣耀就够了
· 苏宁云商称已出售PPTV股权至公司实控人张近东的企业
· 微软云服务又宕机了，这是本月第2次
· 腾讯市值1.86万亿元 力压阿里成市值最高新兴市场公司
» 更多新闻...

最新知识库文章:
· 程序员学习能力提升三要素
· 为什么我要写自己的框架?
· 垃圾回收原来是这么回事
· 「代码家」的学习过程和学习经验分享
· 写给未来的程序媛
» 更多知识库文章...