

基于 ZooKeeper 的配置信息存储方案的设计与实现

王阿晶, 邹仕洪

(北京邮电大学网络与交换技术国家重点实验室, 北京 100876)

摘要: 随着云计算的兴起, 分布式计算成为研究的热点, 而分布式应用的配置信息的管理也变得非常重要。本文提出了一种基于 ZooKeeper 的配置信息存储方案。首先介绍了 ZooKeeper 的架构和 ZooKeeper 的相关概念, 然后分析了当前配置信息存储方案的不足; 最后重点介绍基于 ZooKeeper 的配置信息存储方案, 包括架构和实现方案等。

关键词: 分布式计算; ZooKeeper; 配置信息存储; Hadoop

中图分类号: TP311

A DESIGN AND IMPLEMENTATION OF CONFIGURATION INFORMATION STORAGE BASED ON ZOOKEEPER

WANG Ajing, ZOU Shihong

(State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876)

Abstract: With the rise of cloud computing, distributed computing has become a hot research, and distributed application configuration information management has become very important. This paper presents a configuration information storage based on ZooKeeper. First introduces ZooKeeper architecture and related concepts, and then analyzes deficiencies of the current configuration information storage; finally, highlights the design and implement of configuration information storage on ZooKeeper, including architecture and implementation of programs.

Keywords: Distributed Computing; ZooKeeper; Configuration Information Storage; Hadoop

0 引言

随着计算机技术的广泛应用和数据量的快速增加, 处理海量数据的分布式计算变得越来越重要。与传统的单机上运行程序不同, 分布式计算将计算分布到多台机器上, 使多台机器同时处理, 从而提高计算效率。目前, Hadoop^[1]作为开源的分布计算框架得到了各大公司的应用。Hadoop 借助 Google 提出的分布式文件系统和 MapReduce, 解决分布式计算的存储和计算框架两大难题, 从而使分布式计算的应用程序开发起来更加容易。

然而要管理这些分布在多台计算机上的应用程序却并不容易。试想分布在超过 100 台的分布式应用正在运行, 这时需要修改运行程序的配置信息, 例如日志级别、业务规则等。要实现这样的功能, 且做到可靠、实时、简单, 具有一定的挑战性。

1 ZooKeeper 简介

ZooKeeper 是 Apache 开源组织下的一个工程, 旨在提供一个简单、可依赖、高性能的分布式应用程序通用的协同服务, 包括命名服务、配置管理服务、同步和组服务^[2]。

Zookeeper 由两部分组成, 分别是 Zookeeper 服务器和客户端。Zookeeper 服务器采用集

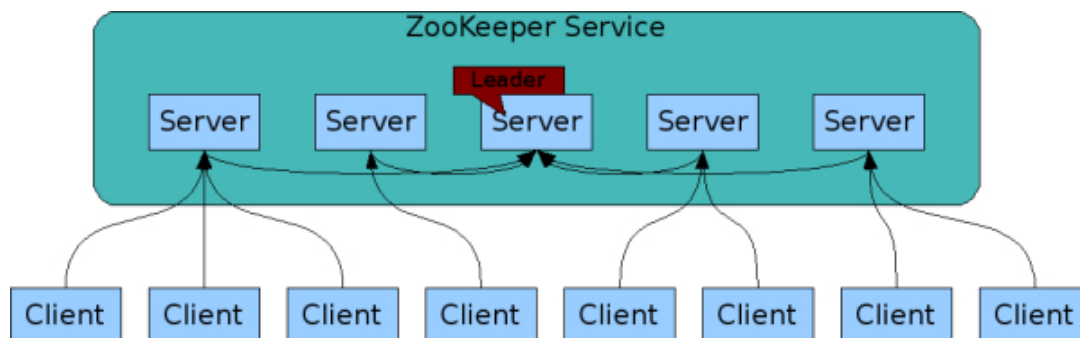
作者简介: 王阿晶, (1988-), 男, 研究生, 主要研究方向: 分布式计算

通信联系人: 邹仕洪, (1978-), 男, 副教授, 主要研究方向包括移动无线网络, 移动信息安全, 网络性能评估与服务质量. E-mail: zoush@bupt.edu.cn

40 群的形式，不存在传统服务器集群的单个失败问题。Zookeeper 集群节点之间进行数据和状态同步。从而保证在任意一台节点失败后，其他节点能够正常工作。客户端提供了 C 和 Java 两种语言访问服务器的 API。通过这些 API，客户端程序可以创建、访问、删除数据，并在数据更新时得到通知，以修改本地程序的状态。

1.1 ZooKeeper 架构

45 ZooKeeper 整体架构^[3]如图 1 所示。在 ZooKeeper 服务集群中，每个服务器都是知道彼此的存在的。客户端在连接 ZooKeeper 服务集群时，客户端会按照一定的随机算法连接到其中一个服务器上。当修改其中一个客户端修改数据时，ZooKeeper 会将修改同步到所有的服务器上，从而使连接到其他服务器上的客户端也能看到数据的修改。当其中一个服务器失效时，客户端会自动会重新选择一个服务器连接，从而保证服务的连续性。



50 图 1 ZooKeeper 整体架构

1.2 ZooKeeper 数据模型

55 Zookeeper 的数据模型采用类似于文件系统的树结构。树上的每个节点称为 ZNode，而每个节点都可能有一个或者多个子节点。不同于文件系统的目录概念，在 Zookeeper 的数据模型中，每个节点都可以保存数据。同时也可以有子节点。

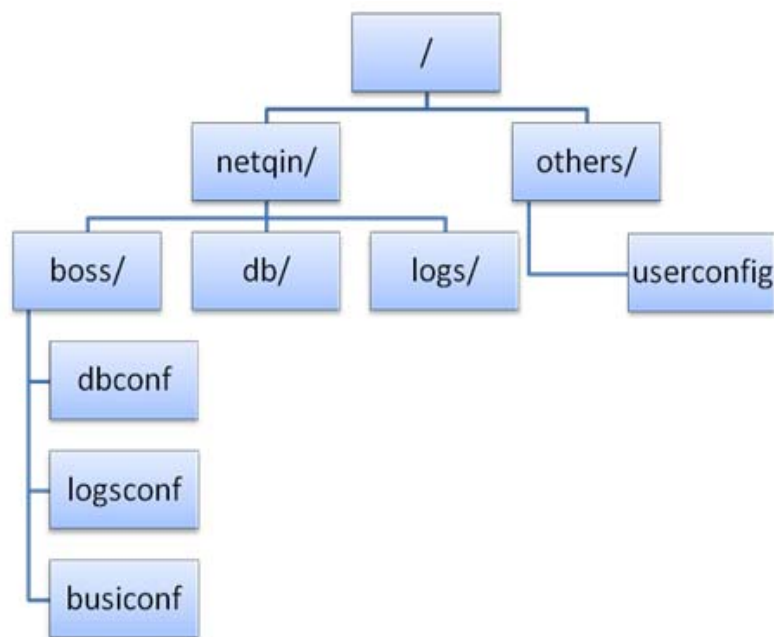


图 2 ZooKeeper 数据模型

60 Znode 除了包含数据之外，还有一个统计结构。统计结构记录了 ZNode 的数据版本，

创建日期，子节点个数，修改日期等详细信息。具体的包含的字段及对应的含义参见表 1。

表 1 Znode 统计结构

字段名称	字段含义
czxid	ZNode 创建时的事务 ID
mzxid	ZNode 最后修改时的事务 ID
ctime	节点创建时间（毫秒数）
mtime	节点修改时间（毫秒数）
version	节点数据的次数
cversion	子节点数据修改的次数
aversion	ZNode 节点访问控制列表的修改次数
ephemeralOwner	如果这个节点是临时节点，该字段表示节点创建者的 sessionId，否则为 0
Datalength	数据长度
numChildren	ZNode 子节点的个数

65 ZooKeeper 对节点数据的读取、写入、更新都是原子操作，从而保证了事务的一致性。每一个节点都有个访问控制列表（ACL），据此控制该节点的访问权限。需要注意的是，ZooKeeper 的设计目标不是传统的数据库存储或者大数据对象存储，而是协同数据的存储。因此 ZNode 存储的数据相对来说是较小的（多是以 KB 为单位），在实现时，ZNode 存储的数据大小不应超过 1MB，否则会降低 ZooKeeper 服务器的性能^[4]。

70 ZooKeeper 的 ZNode 有两种类型，一种是临时性的（EPHEMERAL），即在创建者与 ZooKeeper 服务器会话断开后，ZNode 节点即被删除；另一种则是永久性的（PERSISTENT），即只有调用删除节点的 API 时，该节点才会被删除。两种类型的 ZNode 节点各有用处。使用临时性节点，可以动态的查看连接客户端的应用是否失败；而使用永久性节点，则可以存储一些共享的配置信息等。

75 1.3 ZNode 数据变化通知

当服务器上的 ZNode 节点的数据发生变化时，如创建、更新、删除节点等，客户端通过可以设置观察者监听数据的变化，得到数据变化的通知。在收到通知时，客户端可以做相应的处理，从而实现特定的功能。

2 ZooKeeper 客户端主要操作

80 ZooKeeper 为 Java 应用程序和 C 应用程序提供了客户端 API。本节以 Java 客户端 API 为例，介绍 ZooKeeper 提供给客户端的功能和操作^[5]。

2.1 创建 ZooKeeper 客户端

```
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher);
```

85 connectString 包含了应包含 ZooKeeper 服务器的地址和端口，对于多个服务器，需要使用逗号来隔开。如“zookeeper-1:2181,zookeeper-2:2181”。sessionTimeout 表示会话的超期时间。因为在创建 ZooKeeper 客户端是异步的，当连接建立成功后，会调用 watcher 的 process 方法，客户端需要实现 watcher 的 process 方法，进行相应的处理。

2.2 创建 ZNode

```
public String create(String path, byte[] data, List<ACL> acl, CreateMode createMode);
```

90 public void create(String path, byte[] data, List<ACL> acl, CreateMode createMode, AsyncCallback.StringCallback cb, Object ctx);

上面的两个方法分别是创建 ZNode 的同步版本和异步版本。第一个方法，创建成功后

返回创建节点的实际路径；而第二个方法调用后立即返回。创建成功后，会回调 `cb` 的 `processResult` 方法。参数 `path` 表示创建节点的路径，`data` 是 `Znode` 的节点数据，`acl` 是访问控制列表。而 `createMode` 表示节点的类型。

以下操作都包含了异步和同步的方法，为方便起见，以下在描述时，只描述同步版本的方法。

2.3 删除 ZNode

```
delete(String path, int version);
```

在删除时，需要指定删除节点的路径和版本。当版本和实际节点相匹配时，该节点才会被删除，否则会抛出异常；如果 `version` 为 -1，则无需进行版本匹配。如果删除的节点包含子节点，则会抛出 `KeeperException.NotEmpty` 异常。

2.4 节点是否存在

```
public Stat exists(String path, Watcher watcher);
```

判断节点 `path` 是否存在，如果存在会返回该节点的数据信息。如果 `watcher` 不为 `null`，则会为该节点设置一个观察器。当该节点被创建、删除、更新时，会调用 `watcher` 的 `process` 方法。需要注意的 `watcher` 是一次性触发器。即当设置 `watcher` 后，第一次的节点变化才会调用 `watcher` 的 `process` 方法。在 `getData`、`getChildren` 方法中也可以设置类似于 `exists` 的 `watcher` 对象。

2.5 获取节点数据

```
public byte[] getData(String path, Watcher watcher, Stat stat);
```

根据 `path` 获取节点的数据。如果获取成功，该方法会将该节点的其他信息填充到 `stat` 中。

2.6 设置节点数据

```
public Stat setData(String path, byte[] data, int version);
```

设置 `path` 对应 `ZNode` 的数据内容为 `data`。`Version` 要修改节点数据的版本。如果不匹配，该方法会抛出异常。

2.7 获取子节点列表

```
List<String> getChildren(String path, Watcher watcher, Stat stat)
```

根据 `path` 获取该 `path` 对应下节点的子节点路径列表。如果获取成功，该方法会将 `path` 对应的节点的其他统计信息填充到 `stat` 中。

3 现有配置信息存储方案分析

在信息大爆炸的今天，单机程序或者单服务器程序显然无法应付海量数据的处理和分析。因此将程序分布式化成了重要的课题。然而程序的分布式化面临的一个最严重的问题是分布式程序的通信和统一管理问题。试想一个分布在上百台机器上的应用程序，如果要修改应用程序的配置，例如日志级别、业务参数配置，目前有以下几种方案：

1. 将配置信息保存在程序代码中

这种方案简单，但每次修改配置都要重新编译、部署应用程序。显然这种方案很不方便，也不可靠，更无法做到修改的实时生效。

130 2. 将配置信息保存在 xml 文件或者属性文件中

在参数信息保存在 xml 或者属性文件中，当需要修改参数时，直接修改 xml 文件。这样无需重新编译，只需重新部署修改的文件即可。但然后对所有的应用进行重新部署。这样做的缺点显而易见，要往上百台机器上重新部署应用，简直是一个噩梦。同时该方案还有一个缺点，就是配置修改无法做到实时生效。修改后往往过一段时间才能生效。

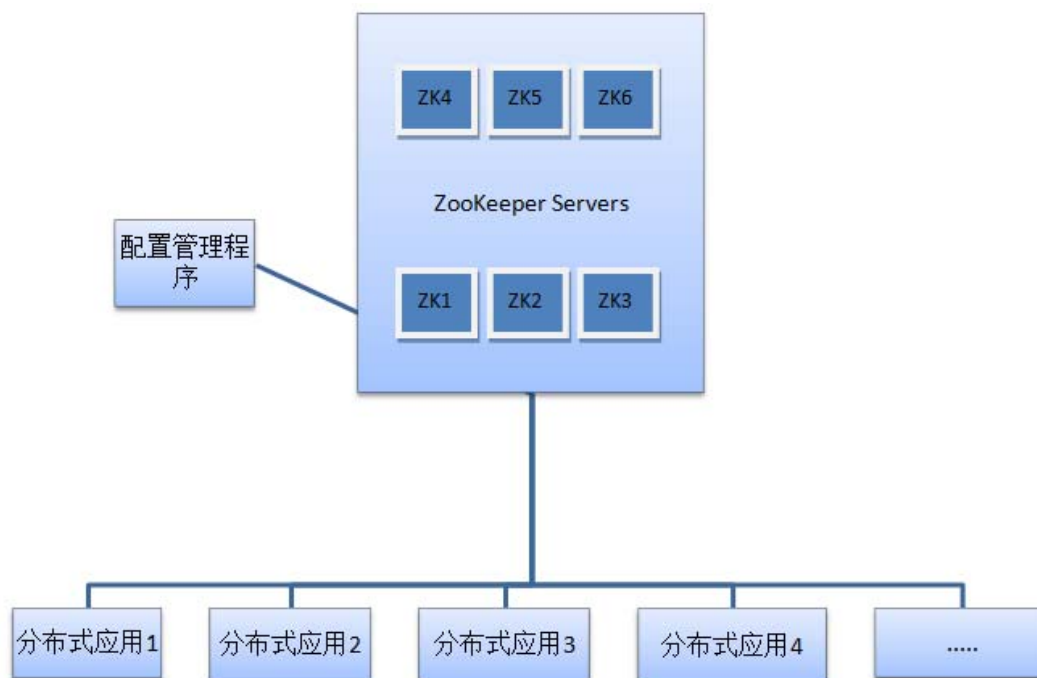
135 3. 将配置信息保存在数据库中

当需要修改参数时，直接修改数据库，然后重启分布式应用程序，或者刷新分布式应用的缓存。尽管这种做法比以上两种方案简单，但却面临着单点失效问题。如果数据库服务器停机，则分布式应用程序的配置信息将无法更新。另外这种方案的配置修改生效实时性虽然比第二种方案好些，但仍然不能达到某些情况下的要求。

140 经分析，尽管上述方案能够工作，但却有着诸多缺点。因此需要提出一个可靠的、简单的、修改配置实时生效的的配置信息存储方案。

4 基于 ZooKeeper 的配置信息存储方案

基于 ZooKeeper 的特性，借助 ZooKeeper 可以实现一个可靠的、简单的、修改配置能够实时生效的配置信息存储方案，整体的设计方案如图 4 所示：

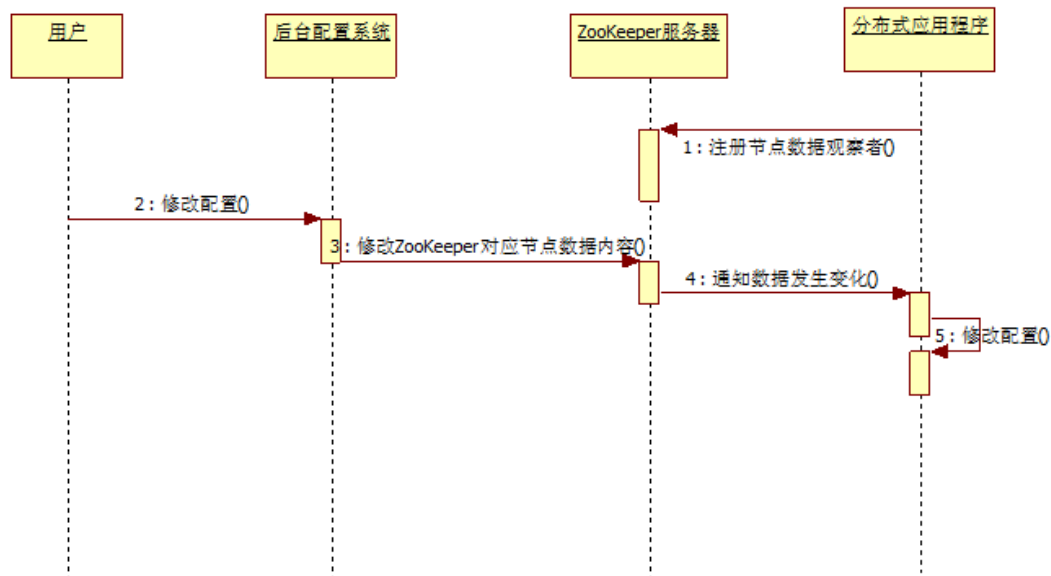


145

图 3 基于 ZooKeeper 的配置信息存储方案架构

整个配置信息存储方案由三部分组成：ZooKeeper 服务器集群、配置管理程序、分布式应用程序。ZooKeeper 服务器集群存储配置信息，在服务器上创建一个保存数据的节点（创建节点操作）；配置管理程序提供一个配置管理的 UI 界面，用户通过配置界面修改 ZooKeeper 服务器节点上配置信息（设置节点数据操作）；分布式应用连接到 ZooKeeper 集群上（创建 ZooKeeper 客户端操作），监听配置信息的变化（使用获取节点数据操作，并注册一个 watcher）。当配置信息发生变化时，分布式应用会更新程序中使用配置信息。

修改数据的时列图如图 4 所示：



155

图 4 修改配置数据的时序图

借助 ZooKeeper 我们实现的配置信息存储方案具有的优点如下：

1. 简单。尽管前期搭建 ZooKeeper 服务器集群较为麻烦，但是实现该方案后，修改配置
160 置整个过程变得简单很多。用户只要修改配置，无需进行其他任何操作，配置自动生效。

2. 可靠。ZooKeeper 服务集群具有无单点失效的特性，使整个系统更加可靠。即使
ZooKeeper 集群中的一台机器失效，也不会影响整体服务，更不会影响分布式应用配置信息
的更新。

3. 实时。ZooKeeper 的数据更新通知机制，可以在数据发生变化后，立即通知给分布式
165 应用程序，具有很强的变化响应能力。

5 结论

本文给出了基于 ZooKeeper 的配置信息存储方案，解决了传统配置信息存储方案的缺点
如实时性差、可靠性差、复杂等。借助该存储方案，可以提高分布式应用程序的快速响应变
化的能力。在实际的项目中，除了针对所有配置修改之外，有时只针对局部的分布式应用的
170 配置进行修改。此时可以使用 ZooKeeper 的子节点，为每个分布式应用创建一个 ZNode，修
改时只修改对应的 ZNode 节点数据即可。

[参考文献] (References)

175 [1] Apache Hadoop[OL]. <http://hadoop.apache.org>
 [2] ZooKeeper[OL]. <http://zookeeper.apache.org/>
 [3] ZooKeeper Overview[OL]. <http://zookeeper.apache.org/doc/r3.3.3/zookeeperOver.html>
 [4] ZooKeeper Programmer's Guide[OL]. <http://zookeeper.apache.org/doc/r3.3.3/zookeeperProgrammers.html>
 [5] ZooKeeper Java API[OL]. <http://zookeeper.apache.org/doc/r3.3.3/api/index.html>