Architecture design | distributed system scheduling, zookeeper cluster management

Time：2020-6-18

Source code: GitHub, click here, gitee, click here

1、 Framework introduction

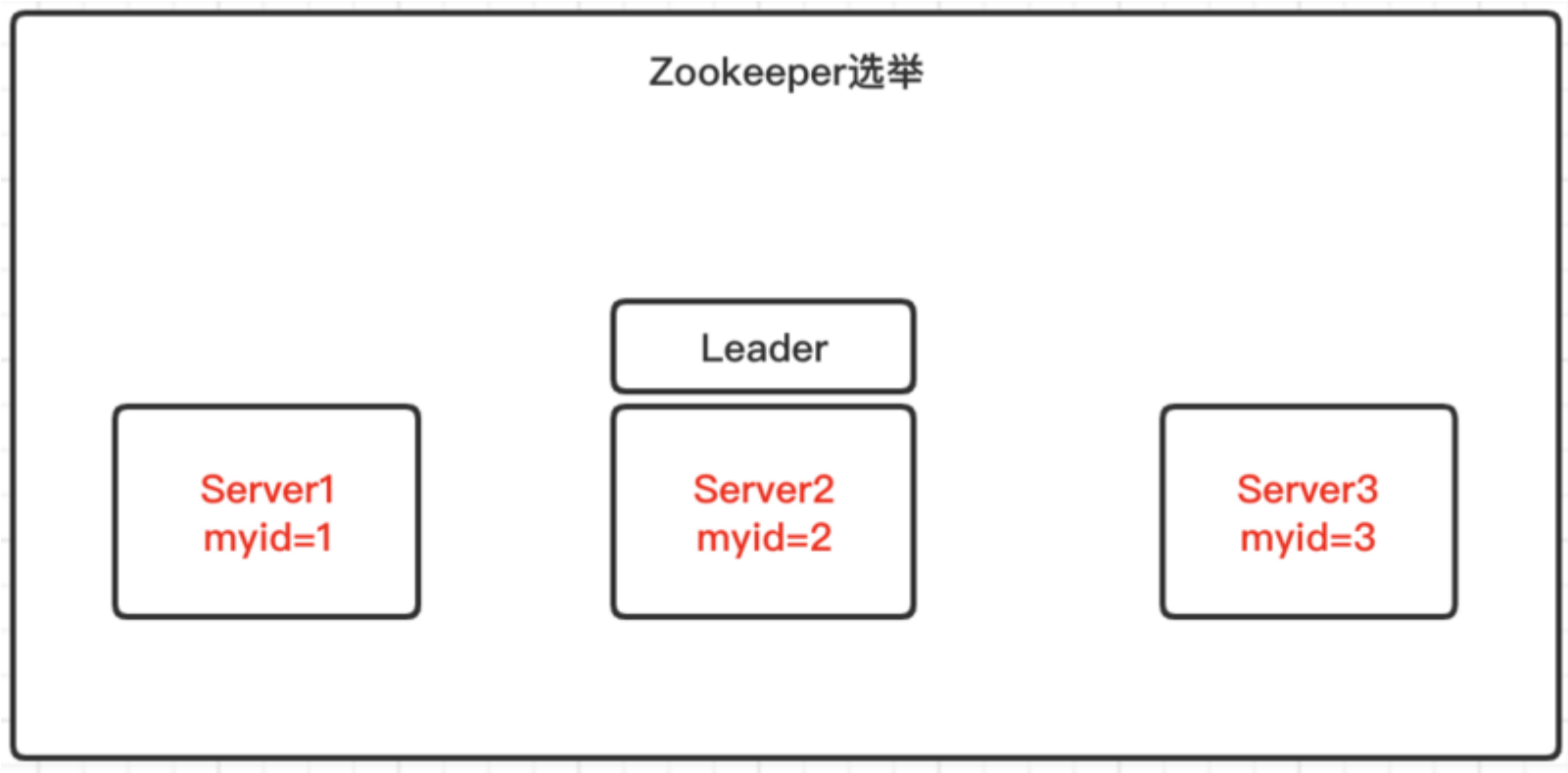
1. Basic introduction

The components designed by zookeeper based on observer mode are mainly used in distributed system architecture, such as unified naming ser vice, unified configuration management, unified cluster management, server node dynamic up and down line, soft load balancing and other scena rios.

- Single node installation of zookeeper under Linux
- Springboot integrates zookeeper Middleware

2. Cluster election

Zookeeper cluster is based on the half mechanism. More than half of the machines in the cluster survive and the cluster is available. Therefore, it is recommended that zookeeper cluster be installed as an odd number of servers. Master and slave are not specified in the cluster configuration fi le. When zookeeper works, one node is leader and the other is follower. Leader is temporarily generated through internal election mechanism.



Basic description

Suppose there are three servers in the zookeeper cluster. The myid number of each node is 1-3, and the server is started in turn. It will be found t hat server2 is selected as the leader node.

Server1 starts and performs an election. Server 1 votes for itself. At this time, the server 1 has one vote, less than half (2 votes), the election cann ot be completed, and the server 1 status remains waiting;

Server2 starts and performs another election. Server 1 and server 2 vote for themselves and exchange vote information respectively. Because th e myid of server 2 is larger than that of server 1, server 1 will change the vote to vote for server 2. At this time, there are 0 votes for server 1 and 2 votes for server 2, reaching more than half of the votes. The election is completed. Server 1 is in the following state, and 2 is in the leader state. At this time, the cluster is available, and server 3 is directly in the following state after it is started.

2、 Cluster configuration

1. Create configuration directory

```
# mkdir -p /data/zookeeper/data
# mkdir -p /data/zookeeper/logs
```

2. Basic configuration

```
# vim /opt/zookeeper-3.4.14/conf/zoo.cfg

tickTime=2000
initLimit=10
syncLimit=5
dataDir=/data/zookeeper/data
dataLogDir=/data/zookeeper/logs
clientPort=2181
```

3. Single node configuration

```
# vim /data/zookeeper/data/myid
```

Three node services, write [1,2,3] in myid file respectively

4. Cluster service

In each service zoo.cfg Write the following configuration in the configuration file:

```
server.1=192.168.72.133:2888:3888
server.2=192.168.72.136:2888:3888
server.3=192.168.72.137:2888:3888
```

5. Start cluster

Start three zookeeper services respectively

```
[zookeeper-3.4.14]# bin/zkServer.sh start
Starting zookeeper ... STARTED
```

6. View cluster status

Mode: leader is the master node

Mode: follower is a slave node

```
[zookeeper-3.4.14]# bin/zkServer.sh status
Mode: leader
```

7. Cluster state test

Log in to any client of a service, create a test node, and view it on other services.

```
[zookeeper-3.4.14 bin]# ./zkCli.sh
[zk: 0] create /node-test01 node-test01
Created /node-test01
[zk: 1] get /node-test01
```

Or close the leader node

```
[zookeeper-3.4.14 bin]# ./zkServer.sh stop
```

The node is re elected.

8. Nginx unified management

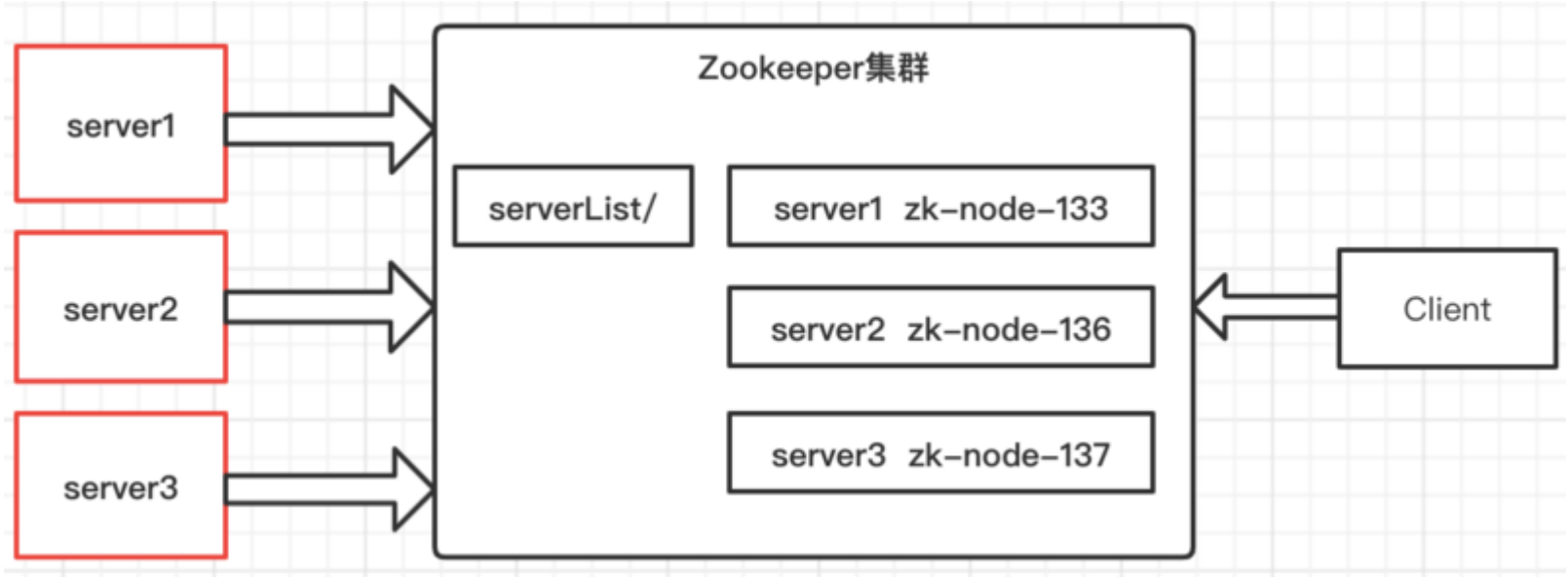
```
[rnginx-1.15.2 conf]# vim nginx.conf

stream {
    upstream zkcluster {
        server 192.168.72.133:2181;
        server 192.168.72.136:2181;
        server 192.168.72.136:2181;
    }
    server {
        listen 2181;
        proxy_pass zkcluster;
    }
}
```

3、 Service node listening

1. Fundamentals

In the distributed system, there can be multiple primary nodes, which can dynamically go up and down the line. Any client can sense the up and d own line of the primary node server in real time.



Process Description:

- Start zookeeper cluster service;
- Registerserver simulates server registration;
- Clientserver simulates client listening;
- Start the server to register three times and register the ZK node services of different nodes;
- Close the registered server in turn and simulate the service offline process;
- View the client log to monitor the changes of service nodes;

First, create a node: serverlist to store the server list.

```
[zk: 0] create /serverList "serverList"
```

2. Server registration

```
package com.zkper.cluster.monitor;
import java.io.IOException;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.ZooDefs.Ids;

public class RegisterServer {

    private ZooKeeper zk ;
    private static final String connectString = "127.0.0.133:2181,127.0.0.136:2181,127.0.0.137:2181";
    private static final int sessionTimeout = 3000;
    private static final String parentNode = "/serverList";

    private void getConnect() throws IOException{
        zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {
            @Override
            public void process(WatchedEvent event) {
            }
        });
    }

    private void registerServer(String nodeName) throws Exception{
        String create = zk.create(parentNode + "/server", nodeName.getBytes(),
                                  Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);
        System.out.println (nodeName + "go online:" + create);
    }

    private void working() throws Exception{

        Thread.sleep(Long.MAX_VALUE);
    }

    public static void main(String[] args) throws Exception {
```

```
        RegisterServer server = new RegisterServer();
        server.getConnect();
        //Start the service three times, register different nodes, and shut down different servers again to see the client effect
        // server.registerServer("zk-node-133");
        // server.registerServer("zk-node-136");
        server.registerServer("zk-node-137");
        server.working();
    }
}
```

3. Client listening

```
package com.zkper.cluster.monitor;
import org.apache.zookeeper.*;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class ClientServer {
    private ZooKeeper zk ;
    private static final String connectString = "127.0.0.133:2181,127.0.0.136:2181,127.0.0.137:2181";
    private static final int sessionTimeout = 3000;
    private static final String parentNode = "/serverList";

    private void getConnect() throws IOException {
        zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {
            @Override
            public void process(WatchedEvent event) {
                try {
                    //Monitor online service list
                    getServerList();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    private void getServerList() throws Exception {
        List<String> children = zk.getChildren(parentNode, true);
        List<String> servers = new ArrayList<>();
        for (String child : children) {
            byte[] data = zk.getData(parentNode + "/" + child, false, null);
            servers.add(new String(data));
        }
        System.out.println ("current service list:" + servers);
    }
```

```
    private void working() throws Exception{
        Thread.sleep(Long.MAX_VALUE);
    }

    public static void main(String[] args) throws Exception {
        ClientServer client = new ClientServer();
        client.getConnect();
        client.getServerList();
        client.working();
    }

}
```

4、 Source code address

```
GitHub · address
https://github.com/cicadasmile/data-manage-parent
Gitee · address
https://gitee.com/cicadasmile/data-manage-parent
```

Recommended reading: data and architecture management

No	title
A01	Data source management: master-slave dynamic routing, AOP mode read-write separation
A02	Data source management: adapt and manage dynamic data sources based on JDBC mode
A03	Data source management: dynamic permission verification, table structure and data migration process
A04	Data source management: relational database, tabular database and distributed computing
A05	Data source management: PostgreSQL environment integration, JSON type application
A06	Data source management: Based on dataX components, synchronous data and source code analysis

No	title
C01	Architecture basis: single service, cluster, distribution, basic difference and connection
C02	Architecture design: Global ID generation strategy in distributed business system

Tags: Client, colony, data source, node, The server