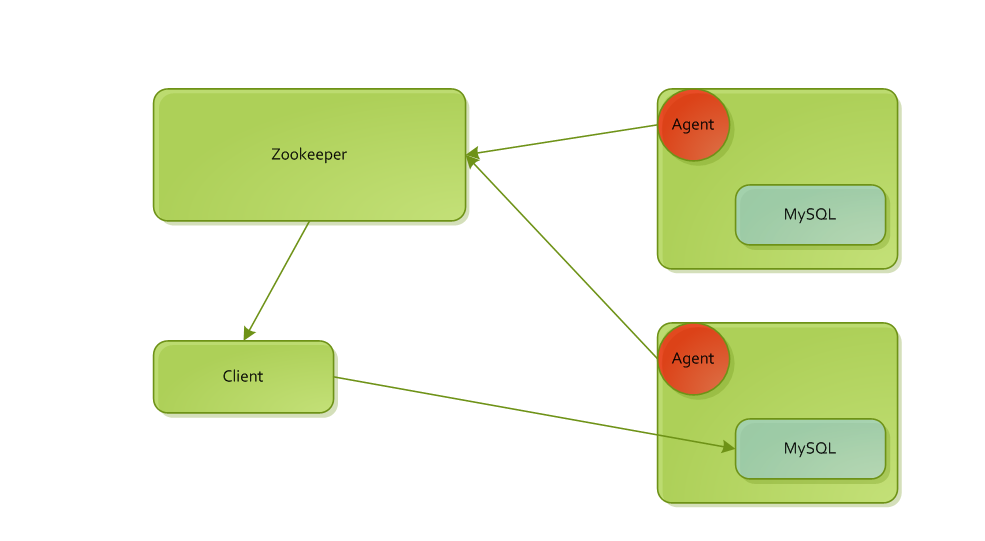
[**基于zookeeper的MySQL主主负载均衡的简单实现**](http://www.cnblogs.com/TomSnail/p/4389297.html)

1.先上原理图



2.说明

两个mysql采用主主同步的方式进行部署。

在安装mysql的服务器上安装客户端（目前是这么做，以后想在zookeeper扩展集成），客户端实时监控mysql应用的可用性，可用时想zookeepercreateNode，当网络不可用或者mysql应用不可用时，建立的znode消失。

在客户端，通过改造proxool数据库连接池的方式，在建立连接之前，从zookeeper中去取真实的数据库URL，如果有多个URL，即有多个服务时，采用随机算法去拿连接（以后准备扩展权重）。当连接不可用时，数据库连接池将重建连接，这时候又回去zookeeper拿连接，因为agent建立的临时znode消失了，就不能拿到已经失效的url了。

这个方案只是初步的实验和实现了，还有很多后续的问题，主要为了解决lvs+keepalived只能在同一个区域内的问题。

3.部分实现

　　1).agent

[复制代码](javascript:void(0);)

/\*\*

\* 数据库可用性检测

\* @author tomsnail

\* @date 2015年4月3日 上午10:11:51

\*/

public class TestMySQL {

public static boolean test(String url){

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

String sql = ConfigHelp.getLocalConifg("jdbc\_inventory.house-keeping-test-sql", "select 0");

try {

Class.forName(ConfigHelp.getLocalConifg("jdbc\_inventory.driver-class", "com.mysql.jdbc.Driver"));// 动态加载mysql驱动

conn = DriverManager.getConnection(url);

stmt = conn.createStatement();

rs = stmt.executeQuery(sql);

while (rs.next()) {

}

return true;

} catch (SQLException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if(rs!=null){

rs.close();

}

if(stmt!=null){

stmt.close();

}

if(conn!=null)

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return false;

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

/\*\*

\* zookeeper客户端

\* @author tomsnail

\* @date 2015年4月3日 上午10:11:51

\*/

public class TestServer {

private static final Logger logger = LoggerFactory

.getLogger(TestServer.class);

private static ZooKeeper zk;

private String path;

//同步锁

private Lock \_lock = new ReentrantLock();

// 用于等待 SyncConnected 事件触发后继续执行当前线程

private CountDownLatch latch = new CountDownLatch(1);

public TestServer() {

zk = connectServer();

new Thread(new Runnable() {

@Override

public void run() {

while (true) {

try {

Thread.currentThread().sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

//logger.info("check zk...");

\_lock.lock();

if (zk != null) {

if (zk.getState().isAlive()

&& zk.getState().isConnected()) {

//logger.info("zk is ok");

\_lock.unlock();

continue;

}

}

close();

logger.info("reConnectServer ...");

zk = connectServer();

logger.info("reConnectServer ok");

\_lock.unlock();

}

}

private void close() {

if(zk!=null){

try {

zk.close();

} catch (InterruptedException e) {

e.printStackTrace();

}

zk = null;

}

}

}).start();

}

// 连接 ZooKeeper 服务器

private ZooKeeper connectServer() {

ZooKeeper zk = null;

try {

zk = new ZooKeeper(ConfigHelp.ZK\_CONNECTION\_STRING,

ConfigHelp.ZK\_SESSION\_TIMEOUT, new Watcher() {

@Override

public void process(WatchedEvent event) {

if (event.getState() == Event.KeeperState.SyncConnected) {

latch.countDown(); // 唤醒当前正在执行的线程

}

}

});

latch.await(); // 使当前线程处于等待状态

} catch (Exception e) {

logger.error("", e);

}

if (zk != null) {

try {

Stat stat = zk.exists(ConfigHelp.ZK\_ROOT\_PATH, false);

if (stat == null) {

String path = zk.create(ConfigHelp.ZK\_ROOT\_PATH,

"".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE,

CreateMode.PERSISTENT); // 创建一个临时性且有序的 ZNode

logger.info("create zookeeper node ({})", path);

}

stat = zk.exists(ConfigHelp.ZK\_RMI\_PATH, false);

if (stat == null) {

String path = zk.create(ConfigHelp.ZK\_RMI\_PATH,

"".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE,

CreateMode.PERSISTENT); // 创建一个临时性且有序的 ZNode

logger.info("create zookeeper node ({})", path);

}

} catch (Exception e) {

e.printStackTrace();

}

}

return zk;

}

// 创建 ZNode

public void createNode(String url) {

\_lock.lock();

try {

byte[] data = url.getBytes();

path = zk.create(ConfigHelp.ZK\_RMI\_PATH + "/", data,

ZooDefs.Ids.OPEN\_ACL\_UNSAFE,

CreateMode.EPHEMERAL\_SEQUENTIAL); // 创建一个临时性且有序的 ZNode

logger.info("create zookeeper node ({} => {})", path, url);

} catch (Exception e) {

logger.error("", e);

e.printStackTrace();

}

\_lock.unlock();

}

public void deleteNode(String url){

\_lock.lock();

try {

Stat stat = zk.exists(path, false);

if(stat!=null){

zk.delete(url, stat.getVersion());

}

} catch (Exception e) {

e.printStackTrace();

}

\_lock.unlock();

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

/\*\*

\* 数据库检测测试主类

\* @author tomsnail

\* @date 2015年4月3日 上午10:11:51

\*/

public class TestMain {

private static TestServer testServer = new TestServer();

public static void main(String[] args) {

String url = ConfigHelp.getLocalConifg("jdbc\_inventory.driver-url", "select 0");

boolean isOK = false;

while(true){

if(TestMySQL.test(url)){

if(isOK){

}else{

testServer.createNode(url);//建立znode

}

isOK = true;

}else{

isOK = false;

testServer.deleteNode(url);//删除znode

}

try {

Thread.currentThread().sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

[复制代码](javascript:void(0);)

　　2).proxool

[复制代码](javascript:void(0);)

/\*\*

\* zookeeper信息定义类

\* @author tomsnail

\* @date 2015年4月2日 下午6:49:13

\*/

public class ZkInfoDefinition {

public static final String PREFIX\_ZK = "zookeeper";

public static final String ZK\_URL = "zkUrl";

public static final String ZK\_SESSION\_TIMEOUT = "sessionTimeout";

public static final String ZK\_PATH = "zkPath";

public static final String ZK\_ENABLE = "zkEnable";

public static String zkUrl="192.168.102.1:31315";

public static int sessionTimeout = 5000;

public static boolean isEnable = false;

public static String zkPath = "/root/db";

public String getZkUrl() {

return zkUrl;

}

public void setZkUrl(String zkUrl) {

this.zkUrl = zkUrl;

}

public int getSessionTimeout() {

return sessionTimeout;

}

public void setSessionTimeout(int sessionTimeout) {

this.sessionTimeout = sessionTimeout;

}

public String getZkPath() {

return zkPath;

}

public void setZkPath(String zkPath) {

this.zkPath = zkPath;

}

public ZkInfoDefinition(String zkUrl, int sessionTimeout, String zkPath) {

super();

this.zkUrl = zkUrl;

this.sessionTimeout = sessionTimeout;

this.zkPath = zkPath;

}

public ZkInfoDefinition(){

}

}

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

/\*\*

\* zookeeper客户端

\* @author tomsnail

\* @date 2015年4月3日 上午10:15:11

\*/

public class ZkClient {

private static final Logger logger = LoggerFactory.getLogger(ZkClient.class);

// 用于等待 SyncConnected 事件触发后继续执行当前线程

private CountDownLatch latch = new CountDownLatch(1);

// 定义一个 volatile 成员变量，用于保存最新的 RMI 地址（考虑到该变量或许会被其它线程所修改，一旦修改后，该变量的值会影响到所有线程）

private volatile List<String> dataList = new ArrayList<String>();

private Lock \_lock = new ReentrantLock();

private static ZooKeeper zk;

private LBUrl lbUrl;

public ZkClient(){

this(new BasicLBUrl());

}

// 构造器

public ZkClient(LBUrl lbUrl) {

this.lbUrl = lbUrl;

zk = connectServer(); // 连接 ZooKeeper 服务器并获取 ZooKeeper 对象

watchNode();

new Thread(new Runnable() {

@Override

public void run() {

while (true) {

try {

Thread.currentThread().sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

\_lock.lock();

if (zk != null) {

if (zk.getState().isAlive()

&& zk.getState().isConnected()) {

\_lock.unlock();

continue;

}

}

if(zk!=null){

try {

zk.close();

} catch (InterruptedException e) {

e.printStackTrace();

}

zk = null;

}

zk = connectServer();

\_lock.unlock();

}

}

}).start();

}

// 查找 URL 服务

public String getUrl() {

if (dataList!=null&&dataList.size()>0) {

return this.lbUrl.getUrl(dataList);

}

return null;

}

public List<String> getUrls(){

return dataList;

}

// 连接 ZooKeeper 服务器

private ZooKeeper connectServer() {

ZooKeeper zk = null;

try {

zk = new ZooKeeper(ZkInfoDefinition.zkUrl, ZkInfoDefinition.sessionTimeout, new Watcher() {

@Override

public void process(WatchedEvent event) {

if (event.getState() == Event.KeeperState.SyncConnected) {

latch.countDown(); // 唤醒当前正在执行的线程

}

}

});

latch.await(); // 使当前线程处于等待状态

} catch (Exception e) {

logger.error("", e);

}

return zk;

}

// 观察 /registry 节点下所有子节点是否有变化

private void watchNode() {

\_lock.lock();

if(zk!=null&&zk.getState().isAlive()&&zk.getState().isConnected()){

}else{

if(zk!=null){

try {

zk.close();

} catch (InterruptedException e) {

e.printStackTrace();

}

zk = null;

}

zk = connectServer();

}

try {

List<String> nodeList = zk.getChildren(ZkInfoDefinition.zkPath, new Watcher() {

@Override

public void process(WatchedEvent event) {

if (event.getType() == Event.EventType.NodeChildrenChanged) {

watchNode(); // 若子节点有变化，则重新调用该方法（为了获取最新子节点中的数据）

}

}

});

List<String> dataList = new ArrayList<String>(); // 用于存放 /registry 所有子节点中的数据

for (String node : nodeList) {

byte[] data = zk.getData(ZkInfoDefinition.zkPath + "/" + node, false, null); // 获取 /registry 的子节点中的数据

dataList.add(new String(data));

}

logger.debug("node data: {}", dataList);

this.dataList = dataList;

} catch (Exception e) {

logger.error("", e);

}

\_lock.unlock();

}

public static void main(String[] args) {

ZkClient client = new ZkClient();

System.out.println(client.getUrl());

}

}

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

/\*\*

\* 从zookeeper获得URL连接操作类

\* @author tomsnail

\* @date 2015年4月2日 下午6:56:06

\*/

public class ZkUrlOperation {

private static final ZkUrlOperation instance = new ZkUrlOperation();

private static ZkInfoDefinition zkInfoDefinition;

private static ZkClient zkClient;

private static final byte[] \_lock = new byte[0];

private ZkUrlOperation(){

}

public static ZkUrlOperation getInstance(){

return instance;

}

public void addZkInfoDefinition(ZkInfoDefinition zkInfoDefinition){

ZkUrlOperation.zkInfoDefinition = zkInfoDefinition;

}

public void addZkInfoDefinition(String key,String value){

if(ZkUrlOperation.zkInfoDefinition==null){

ZkUrlOperation.zkInfoDefinition = new ZkInfoDefinition();

}

if(key.contains(ZkInfoDefinition.ZK\_PATH)){

ZkUrlOperation.zkInfoDefinition.setZkPath(value);

}

if(key.contains(ZkInfoDefinition.ZK\_SESSION\_TIMEOUT)){

ZkUrlOperation.zkInfoDefinition.setSessionTimeout(Integer.valueOf(value));;

}

if(key.contains(ZkInfoDefinition.ZK\_URL)){

ZkUrlOperation.zkInfoDefinition.setZkUrl(value);;

}

if(key.contains(ZkInfoDefinition.ZK\_ENABLE)){

ZkUrlOperation.zkInfoDefinition.isEnable = Boolean.valueOf(value);

}

}

public String getUrl(){

synchronized (\_lock) {

if(zkInfoDefinition.isEnable){

if(zkClient==null){

zkClient = new ZkClient();

}

String url = zkClient.getUrl();

return url;

}else{

return "";

}

}

}

public boolean isAvailUrl(String url){

synchronized (\_lock) {

if(zkInfoDefinition.isEnable){

if(zkClient==null){

zkClient = new ZkClient();

}

List<String> urls = zkClient.getUrls();

for(int i=0;i<urls.size();i++){

if(url.equals(urls.get(i))){

return true;

}

}

return false;

}

return false;

}

}

}

[复制代码](javascript:void(0);)